微型计算机原理与汇编语言

●潘 峰 主编



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.co.cn

微型计算机原理与汇编语言

潘峰　主编

吕建平　王宣政　王钰　彭家敏　参编

（排名无序）

電子工業出版社

Publishing House of Electronics Industry

内 容 简 介

本书是参照国家教委工科计算机基础课教学基本要求的精神和全国非计算机专业等级(3 级 A)考试大纲的要求编写的。书中以 8086 机型为背景，详细介绍了微机系统的基本组成、工作原理和实际应用，并适当介绍了 80286、80386、80486 及 Pentium 机的特点，可作为高等学校工科非计算机专业学生学习有关微机系统原理及应用、学习汇编语言程序设计的教材，也可供从事微机硬件或软件工作的工程技术人员进行科研开发时参考。本书配有相应的实验教材。

书　　名：微型计算机原理与汇编语言

著　　者：潘峰　　主编

责任编辑：李新社

特约编辑：康宗朗

印 刷 者：北京科技大学印刷厂

装 订 者：三河赵华装订厂

出版发行：电子工业出版社出版、发行　北京市海淀区万寿路 173 信箱　邮编 100036

发行部电话 68214070

URL：http://www.phei.co.cn

经　　销：各地新华书店经销

开　　本：787×1092 毫米 1/16　印张：27.75　字数：692.6 千字

版　　次：1997 年 4 月第一版　1997 年 4 月第一次印刷

印　　数：8000 册

书　　号：ISBN 7-5053-4007-7 / G・306

定　　价：28.00 元

凡购买电子工业出版社的图书，如有缺页、倒页、脱页者，本社发行部负责调换

版权所有・翻印必究

前　言

本书是根据编者多年从事高校各专业微机课程的教学实践，并参照国家教委工科计算机基础课程教学基本要求的精神和全国非计算机专业等级（3级A）考试大纲的要求编写的。本书可作为高等学校工科非计算机专业学生学习有关微机系统原理及应用以及学习汇编语言程序设计的教材，也可供从事微机硬件或软件工作的工程技术人员参考。

微型计算机自问世以来，一直以令人目不暇接的态势飞速发展，新机型、新技术、新的应用层出不穷，日新月异。因此，微机课程的教学内容需要不断更新和充实。然而，要使教材及时跟上微机迅速发展的形势是十分困难的。正如国家教委对课程教学基本要求中所指出的：微机课程是工科学生学习和掌握微机硬件知识和汇编语言程序设计的入门课程，课程的任务是使学生从理论和实践上掌握微机的基本组成、工作原理、接口电路及硬件的连接，建立微机系统的整机概念，使学生具有微机系统软硬件开发的初步能力。本书本着上述指导思想，以8086机型为背景，介绍了微机系统的基本组成、工作原理和实际应用，并适当介绍80286、80386、80486及Pentium等微型机的特点，以便使学生掌握基本原理，增强分析问题和解决问题的能力，通过学习举一反三，以适应微机发展形势的需要。由于微机课程是应用技术基础课，教材编写中应注重应用，叙述深入浅出，力求实用，并注意与微机的发展相适应。在教学中，应注意加强实践环节，通过适当的思考题与习题和适量的上机实践，培养学生用微机作为工具进行实验研究的能力及软硬件方面的实际开发能力。

本书第一章和第十章由吕建平编写，第二章和第九章由彭家敏编写，第三章、第七章及附录由潘峰编写，第四章和第五章由王钰编写，第六章和第八章由王宣政编写，潘峰担任主编，负责全书内容的修改和最后定稿。以上编者均长期从事微机课程的教学工作。在本书编写过程中，自始至终得到了西安邮电学院计算机系领导、同行专家的指导、关心、帮助和全力支持。本书的出版过程中得到了电子工业出版社、北京航空航天大学康宗朗老师的帮助和支持，编者在此一并表示衷心的谢意。

由于编者水平有限，加之时间仓促，书中一定存在错误及不妥之处，请专家和读者不吝赐教。

编　者

目　录

第一章　概述……………………………………………………………………………（1）
第一节　微型计算机概述………………………………………………………………（1）
一、微型计算机的发展 ………………………………………………………………（1）
二、微型计算机的特点 ………………………………………………………………（1）
三、微型计算机的分类 ………………………………………………………………（2）
四、微型计算机的应用范围 …………………………………………………………（3）
第二节　微型计算机系统的组成………………………………………………………（4）
一、一般计算机的结构框图 …………………………………………………………（4）
二、微处理器 …………………………………………………………………………（4）
三、微型计算机 ………………………………………………………………………（4）
四、微型计算机系统 …………………………………………………………………（6）
五、微型计算机的开发系统 …………………………………………………………（7）
六、微机结构的特点——总线结构 …………………………………………………（7）
七、软件是微型计算机系统的重要组成部分 ………………………………………（8）
八、微型计算机硬件技术及其发展趋势 ……………………………………………（9）
第三节　微型计算机的工作过程………………………………………………………（11）
第二章　计算机中的信息表示方法…………………………………………………（13）
第一节　数值及其编码…………………………………………………………………（13）
一、无符号数的表示及运算 …………………………………………………………（13）
二、带符号数的表示及运算 …………………………………………………………（17）
三、定点数和浮点数 …………………………………………………………………（21）
四、二进制编码的十进制数(BCD 码) ………………………………………………（23）
第二节　计算机中常用的字符编码……………………………………………………（24）
一、字符编码(ASCII 码) ……………………………………………………………（24）
二、汉字编码(国标码) ………………………………………………………………（25）
思考题与习题……………………………………………………………………………（26）
第三章　80X86 系列微型计算机的体系结构………………………………………（30）
第一节　8086/8088 CPU ……………………………………………………………（30）
一、8086/8088 CPU 的编程结构 ……………………………………………………（30）
二、8086/8088 CPU 的引脚及其功能 ………………………………………………（34）
三、8088 与 8086 的比较 ……………………………………………………………（39）
第二节　8086/8088 系统总线的构成 ………………………………………………（39）
一、最小方式下系统总线的构成 ……………………………………………………（39）
二、最大方式下系统总线的构成 ……………………………………………………（44）
第三节　存储器和 I/O 的组织 ………………………………………………………（52）
一、存储器的组织 ……………………………………………………………………（52）
二、8088/8086 的 I/O 组织 …………………………………………………………（57）

三、80386/80486 系统的存储结构 …………………………………………………… (58)
第四节 86X86 系统的操作和总线周期…………………………………………… (60)
一、系统的复位和启动操作 ……………………………………………………… (60)
二、总线操作 ………………………………………………………………………… (62)
三、最小方式下的总线保持 ……………………………………………………… (68)
四、最大方式下的总线请求/允许……………………………………………… (69)
五、80X86 系统时序介绍 ………………………………………………………… (70)
思考题与习题…………………………………………………………………………… (71)
第四章 指令系统和寻址方式…………………………………………………… (74)
第一节 8086/8088 指令系统的寻址方式 ……………………………………… (74)
一、操作数的种类 …………………………………………………………………… (74)
二、寻址方式 ………………………………………………………………………… (75)
第二节 8086/8088 指令码格式 …………………………………………………… (80)
第三节 8086/8088 指令系统 ……………………………………………………… (82)
一、数据传送指令 …………………………………………………………………… (83)
二、算术运算指令 …………………………………………………………………… (87)
三、位操作指令 ……………………………………………………………………… (103)
四、串操作指令 ……………………………………………………………………… (110)
五、控制转移指令 …………………………………………………………………… (116)
六、处理器控制指令 ………………………………………………………………… (123)
第四节 80X86 扩充的和增加的指令……………………………………………… (125)
一、80286 扩充的和增加的指令 ………………………………………………… (125)
二、80386、80486 扩充的和增加的指令 ……………………………………… (127)
思考题与习题…………………………………………………………………………… (129)
第五章 汇编语言程序设计……………………………………………………… (132)
第一节 汇编语言的基本概念……………………………………………………… (132)
第二节 汇编语言源程序的格式…………………………………………………… (133)
一、分段结构 ………………………………………………………………………… (133)
二、汇编语言语句的类型和格式 ………………………………………………… (133)
第三节 伪指令语句…………………………………………………………………… (139)
一、数据定义伪指令 ………………………………………………………………… (139)
二、符号定义伪指令 ………………………………………………………………… (141)
三、段定义伪指令 …………………………………………………………………… (142)
四、过程定义伪指令 ………………………………………………………………… (146)
五、模块定义与连接伪指令 ……………………………………………………… (147)
第四节 宏指令语句…………………………………………………………………… (148)
一、MACRO/ENDM ………………………………………………………………… (149)
二、PURGE …………………………………………………………………………… (150)
三、宏指令与子程序的区别 ……………………………………………………… (150)
第五节 汇编语言程序的上机过程………………………………………………… (151)

一、用编辑程序建立汇编语言源程序文件(ASM 文件)…………………………………… (151)
二、用汇编程序将 ASM 文件汇编成目标程序文件(OBJ 文件) ………………………… (153)
三、用连接程序生成可执行程序文件(EXE 文件) ………………………………………… (154)
四、程序的执行 …………………………………………………………………………………… (155)
五、汇编语言和操作系统 PC-DOS 的接口 ……………………………………………………… (155)
第六节 汇编语言程序设计的基本方法…………………………………………………………… (156)
一、汇编语言程序设计的基本过程 ……………………………………………………………… (156)
二、程序结构化的概念 …………………………………………………………………………… (157)
三、简单程序设计 ………………………………………………………………………………… (159)
四、分支程序设计 ………………………………………………………………………………… (160)
五、循环程序设计 ………………………………………………………………………………… (163)
六、子程序设计 …………………………………………………………………………………… (170)
七、DOS 系统功能调用 …………………………………………………………………………… (180)
第七节 程序设计举例……………………………………………………………………………… (184)
一、十进制数算术运算 …………………………………………………………………………… (184)
二、代码转换 ……………………………………………………………………………………… (187)
三、表的处理和应用 ……………………………………………………………………………… (190)
思考题与习题……………………………………………………………………………………… (196)
第六章 存储器…………………………………………………………………………………… (199)
第一节 概述……………………………………………………………………………………… (199)
一、半导体存储器的分类 ………………………………………………………………………… (199)
二、半导体存储器的主要技术指标 ……………………………………………………………… (200)
第二节 读写存储器(RAM) ……………………………………………………………………… (200)
一、静态 RAM(SRAM) …………………………………………………………………………… (200)
二、动态 RAM(DRAM) …………………………………………………………………………… (202)
第三节 只读存储器(ROM) ……………………………………………………………………… (204)
一、掩模式 ROM(MROM) ………………………………………………………………………… (204)
二、可编程只读存储器(PROM) …………………………………………………………………… (204)
三、可擦可编程只读存储器(EPROM) …………………………………………………………… (205)
四、电擦写可编程只读存储器(E^2PROM) ……………………………………………………… (208)
第四节 存储器的组织……………………………………………………………………………… (210)
一、存储器的结构 ………………………………………………………………………………… (210)
二、8086 系统的存储器组织 ……………………………………………………………………… (214)
三、80X86 存储系统简介 ………………………………………………………………………… (220)
思考题与习题……………………………………………………………………………………… (230)
第七章 输入和输出系统………………………………………………………………………… (232)
第一节 概述……………………………………………………………………………………… (232)
一、I/O 与接口电路 ……………………………………………………………………………… (232)
二、CPU 与外设间交换的信息 …………………………………………………………………… (232)
三、接口电路的功能 ……………………………………………………………………………… (233)

四、I/O 讨论的问题 …… (234)
第二节 I/O 端口的编址方式 …… (234)
一、I/O 端口地址与内存单元地址统一编址方式 …… (234)
二、I/O 端口与内存独立编址 …… (235)
第三节 输入/输出控制方式 …… (235)
一、直接程序控制的 I/O 方式 …… (235)
二、中断控制的 I/O 方式 …… (241)
三、直接存储器存取(DMA)I/O 方式 …… (241)
四、IOP 方式 …… (244)
第四节 DMA 控制器 Intel 8237 …… (252)
一、8237 的结构和引脚 …… (252)
二、8237 的工作时序 …… (254)
三、8237 的编程 …… (256)
四、8237 的应用 …… (260)
思考题与习题 …… (263)
第八章 中断 …… (267)
第一节 中断原理 …… (267)
一、中断过程 …… (267)
二、中断源的识别 …… (268)
三、中断优先级的确定 …… (269)
第二节 8086 中断系统 …… (271)
一、8086 中断类型 …… (271)
二、8086 的中断处理 …… (272)
三、80386/80486 的中断 …… (275)
第三节 可编程中断控制器 8259A …… (276)
一、8259A 的内部结构及引脚 …… (276)
二、8259A 的中断管理方式 …… (278)
三、8259A 的编程 …… (280)
四、8259A 与 PC 机的硬件中断 …… (287)
第四节 8086 中断矢量表的建立 …… (288)
一、绝对地址置入法 …… (288)
二、使用串送存指令装入法 …… (289)
三、使用 DOS 调用 …… (289)
四、直接装入法 …… (291)
第五节 82380 中断控制器介绍 …… (292)
一、PIC 功能与内部结构 …… (292)
二、82380 PIC 中断响应时序 …… (293)
三、82380 PIC 的操作形态 …… (294)
四、可编程寄存器 …… (297)
五、82380 PIC 初始化编程 …… (298)

思考题与习题…………………………………………………………………… (302)
第九章　输入/输出接口 …………………………………………………………… (304)
第一节　并行数据通信接口技术 ……………………………………………… (304)
一、并行通信与并行接口 ……………………………………………………… (304)
二、可编程并行通信接口芯片 8255A …………………………………………… (305)
第二节　串行数据通信接口技术……………………………………………… (312)
一、串行通信与串行接口 ……………………………………………………… (312)
二、可编程串行通信接口芯片 8251A …………………………………………… (317)
第三节　计数/定时控制器接口技术 ………………………………………… (325)
一、概述 …………………………………………………………………………… (325)
二、可编程计数/定时器芯片 8253 ……………………………………………… (326)
第四节　多功能 I/O 接口芯片 82380 ………………………………………… (331)
一、82380 的结构 ………………………………………………………………… (331)
二、82380 的 DMA 功能 ………………………………………………………… (332)
三、82380 的中断功能 …………………………………………………………… (332)
四、82380 的定时器 ……………………………………………………………… (333)
第五节　数/模(D/A)和模/数(A/D)转换技术及其接口 ……………………… (333)
一、D/A 转换器 …………………………………………………………………… (334)
二、A/D 转换器 …………………………………………………………………… (340)
思考题与习题…………………………………………………………………… (347)
第十章　高性能微处理器……………………………………………………… (351)
第一节　80286 微处理器……………………………………………………… (351)
一、概述 …………………………………………………………………………… (351)
二、80286 的结构 ………………………………………………………………… (351)
三、80286 的引脚 ………………………………………………………………… (354)
四、80286 的实地址方式 ………………………………………………………… (358)
五、80286 的虚地址保护方式 …………………………………………………… (359)
第二节　80386 微处理器……………………………………………………… (369)
一、概述 …………………………………………………………………………… (369)
二、80386 的基本结构 …………………………………………………………… (370)
三、80386 的引脚 ………………………………………………………………… (377)
四、80386 的工作方式 …………………………………………………………… (380)
第三节　80486 微处理器……………………………………………………… (381)
一、概述 …………………………………………………………………………… (381)
二、80486 的基本结构 …………………………………………………………… (383)
三、80486 的引脚 ………………………………………………………………… (388)
四、80486 多处理机基本结构 …………………………………………………… (392)
五、80486 的工作方式 …………………………………………………………… (394)
第四节　Pentium 微处理器 …………………………………………………… (393)
一、概述 …………………………………………………………………………… (393)

二、Pentium 的功能结构 …… (394)
三、Pentium 的引脚 …… (395)
四、Pentium 的主要特点 …… (402)
思考题与习题…… (405)
附录 A　8086/8088 指令对标志位的影响 …… (408)
附录 B　8086/8088 指令编码一览表 …… (409)
附录 C　ASCII 码控制符号的定义 …… (420)
附录 D　系统功能调用一览表…… (421)
附录 E　BIOS 中断 …… (428)
主要参考资料…… (432)

第一章 概 论

第一节 微型计算机概述

一、微型计算机的发展

电子计算机是由各种电子器件组成的能够自动、高速、精确地进行逻辑控制和信息处理的现代化设备。从第一台电子计算机出现至今的40多年来，已大致经历了电子管式计算机、晶体管式计算机、集成电路式(中、小规模)计算机、大规模集成电路计算机等几个阶段。现在世界上许多国家正在加紧研制以人工智能、神经网络为主要特征的新一代计算机。

电子计算机按其性能来分，有巨型、中型、小型和微型计算机。微型计算机的核心部分是微处理器或微处理机，它是指由一片或几片大规模集成电路组成的，具有运算器和控制器功能的中央处理器(CPU)。

以微处理器为核心，配上由大规模集成电路制作的存储器、输入/输出接口电路及系统总线所组成的计算机，简称微型计算机。以微型计算机为中心，配以相应的外围设备、电源和辅助电路，以及指挥微型计算机工作的系统软件，就构成了微型计算机系统。

自从微处理器和微型计算机问世以来，按CPU字长和功能划分，它已经历了五代的演变。

第一代(1971～1973年)是4位和低档8位微机。代表产品是美国Intel公司的4004微处理机及由它组成的MCS—4微型计算机。

第二代(1974～1978年)是中高档8位微机，以Intel公司的8080和8085，Motorola公司的MC6800，美国Zilog公司的Z80等为CPU的微型机为典型代表。

第三代(1978～1981年)是16位微机，如8086，Z8000和MC68000为CPU的微型机。

第四代(1981～1992年)是32位微机，典型的CPU产品有80386，MC68020。之后，Intel公司又推出80486微处理器。

第五代(1993年以后)是64位微机。1993年3月Intel公司推出了当前最先进的微处理器芯片——64位的Pentium，该芯片采用了新的体系结构，其性能大大高于Intel系列的其它微处理器，为处理器体系结构和PC机的性能引入了全新的概念。

当前微型计算机技术正向着生产领域、服务部门和日常生活的各个领域不断渗透，以其很高的性能/价格比，其应用会越来越广泛。

二、微型计算机的特点

建立在微电子技术加工工艺基础上的微型计算机有许多突出的优点，正是由于这些优点，使它从问世以来，就得到了极其迅速的发展和广泛的应用。

1. 功能强

微型计算机的设计，参考了其它类型计算机的优点，与别的电子设备比较，它的运算速度快，计算精度高，具有记忆和逻辑判断能力，而且每种微处理器都配有一整套支持相应微型计算机工作的软件。硬件和软件的配合，相辅相成，使微型计算机的功能大大增强，适合各行各业的各种不同目的的应用。

2. 可靠性高

由于微处理器及其配套系列芯片上可做出几千、几万甚至几百万个元件，这就减少了大量的焊点、连线、接插件等不可靠因素，使可靠性大大增加。据某些资料估计，芯片集成度增加100倍，系统的可靠性也可增加100倍。目前，微处理器及其系列芯片的平均无故障时间可达10^7～10^8小时。

3. 价格低

微处理器及其配套系列芯片采用集成电路工艺，集成度高，适合工厂大批量生产。因此，产品造价十分低廉。据估计集成度增加100倍，其价格也可降为同功能分立元件的百分之一。

很显然，低的价格对于微型计算机的推广和普及是极为有利的。

4. 适应性强

在微型计算机中，硬件扩展是很方便的，而且系统的软件是很容易改变的。因此，在相同的配置情况下，只要对硬件和软件稍作某些变动就能适应不同用户的要求。

5. 周期短，见效快

微处理器制造厂家除生产微处理器芯片外，还生产各种配套的支持芯片，同时也提供许多有关的支持软件，这就为我们构成一个微型计算机应用系统创造了十分有利的条件。从而可以节省研制时间，缩短研制周期，使研制的系统很快地投入运行，取得明显的经济效益。

6. 体积小，重量轻，耗电省

微处理器及其配套支持芯片的尺寸均比较小，最大也不过几百平方毫米。另外，近几年在微型计算机中还大量地采用了ASIC(大规模集成专用芯片)和GAL(通用可编程门阵列)器件，使得微型计算机的体积明显缩小。

目前微型计算机中的芯片大多采用MOS和CMOS工艺，这样耗电量就很少。这对那些在体积、重量、功耗等方面要求比较严格的使用者来说，是很有意义的。一些过去想使用计算机，但又无法实现的领域，例如航空、航天等部门中的某些应用领域，现在利用微型计算机就很容易实现某些设想。

7. 维护方便

现在用微处理器及其系列技术所构成的微型计算机已逐渐趋于标准化、模块化和系列化，从硬件结构到软件配置都作了较全面的考虑。一般都可用自检、诊断及测试发现系统故障；另一方面，发现故障以后，排除故障也比较容易，如可迅速更换标准化模块或芯片。

三、微型计算机的分类

微处理器(MP－MICRO PROCESSOR)是集成在一片大规模集成电路芯片上的中央处理器，又称MPU，简称MP。它具有一般CPU的功能，但它的体积远远小于一般CPU，还具有功耗低、价廉和可靠性高的优点。

按MPU处理数据的位数来看，微处理器可分为4位、8位、16位和32位MPU。32位微处理器是当今较先进最流行的微处理器，它所构成的微型计算机也是当今世界最流行的、较先进的微型计算机。

从制造微处理器器件的工艺来看，可分为MOS工艺的通用微处理器和双极型TTL工艺的位片式微处理器，后者具有速度快、灵活多变、但功耗较大的特点。位片式微处理器以位为单位构成MPU芯片，常用多片位片式微处理构成高速、分布式系统和阵列式系统等，可以按实际需要，选择构成不同位数的微处理器，因而使用灵活多变。

四、微型计算机的应用范围

微型计算机具有价格低廉、体积小、重量轻、功耗低、可靠性高、使用灵活等优点，且随着大规模和超大规模集成电路工艺的不断发展；其功能亦不断增强，使得微型计算机的应用日益深入各行各业，促进了世界新产业革命的到来。微型计算机在当今信息时代是不可缺少的一种重要工具。

微型计算机按其复杂程度的不同，可适用于各种行业，从仪器仪表和家电的智能化，到科学计算、自动控制、数据和事务处理、辅助设计、辅助教学和人工智能等均可应用微型计算机。

低档的微型机和单片式微型机在仪器仪表和家电的智能化方面的应用，取代了过去的硬件逻辑电路对仪器仪表和家电的控制，这种控制本身并不复杂，但却经常是重复的，从而使逻辑电路庞杂。采用微型计算机之后，很容易简化其控制逻辑，用程序的重复执行以及循环控制，可以作到电路最省、控制更佳，并可通过修改程序来修改控制方案，因而灵活多变、可靠性高。

自从 80 年代初期 IMB PC 系列个人计算机出现以后，微型计算机在数据处理和管理方面的应用迅速推广。不单是工矿企业的各种数据和管理（人事档案管理、物资器材管理、合同管理、工资管理、帐务管理、办公室自动化、生产管理、学生学籍管理等等）可用微型计算机，而且微型计算机还深入到家庭的应用。

由于 IBM PC 系列机在我国的汉化成功，计算机在我国各行各业的应用也是日新月异的，特别是在企业管理和办公室自动化等方面的应用，已为各级领导所接受并给以重视。随着功能更完善的微型计算机的不断推出，这种应用也日益深入人心。为在我国普及计算机的应用创造了条件。

微型计算机另一主要应用方面是生产过程的自动控制。比起常规的自动控制仪表而言，微型计算机自动控制系统具有不可比拟的优点：

1. 对生产过程的控制方式是用软件编程来实现的，因而灵活多变。对不同的生产工艺要求只需改变程序即可。

2. 对生产过程的控制算法，可以实现最佳控制，这是常规控制器很难实现的。

3. 微型机控制系统具有较高的实时性，能迅速获得被控制系统的变化情况，迅速根据生产工艺要求进行控制，从而可以提高产品的质量及合格率。

4. 微型机控制系统可以实现多回路、多参数的控制。一个微机控制系统可以极好地替代多台常规控制器进行控制，从而节省经费投资，提高系统的利用率。

5. 可以与其它微型计算机进行横向或纵向的通信联络，从而实现网络控制方式或多级控制方式，提高系统的可靠性，或实现生产过程的全局自动化。

微型计算机在辅助设计（CAD——COMPUTER－AIDED DESIGN）、辅助教学（CAI——COMPUTER－AIDED INSTRUCTION）、辅助生产（CAM——COMPUTER－AIDED MANUFACTURING）等方面的应用，是用功能很强的高挡微机来实现的。它要求具有足够的存储器空间、高分辨率的显示器、丰富的系统软件和应用软件。

微型计算机在科学计算和数据处理方面的应用，是与微处理器单机功能的增强和协处理器的发展分不开的。协处理器是专用微处理器，例如数值数据处理器 INTEL 8087 就是协同 INTEL 8086/8088CPU 处理浮点数的。数据处理能力增强的微型计算机（或多处理器系统）在

科学计算方面能够发挥较大的作用。

微型机的应用非常广泛，在此就不一一列举了。

第二节 微型计算机系统的组成

一、一般计算机的结构框图

如图 1-1 所示，它主要有运算器、控制器、存储器和输入/输出接口四部分组成。

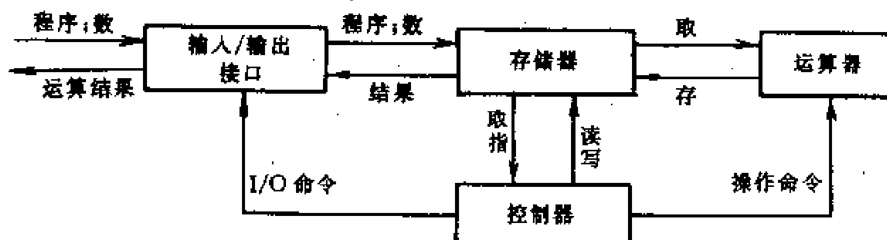


图 1-1 一般计算机结构框图

控制器：发布各种操作命令、控制信号等。

运算器：主要进行算术和逻辑运算。

存储器：存储程序、数据、中间结果和运算结果。

I/O 接口：原始数据和程序等通过输入接口送到存储器，而处理结果、控制信号等通过输出接口送出。

这种以二进制和程序控制为基础的计算机结构是由冯·诺依曼在 1940 年最早提出的。

二、微处理器

微处理器在 1971 年由美国首先研制成功，它将运算器、控制器集成在一片或几片大规模集成电路芯片上。组成了中央处理部件(Centrol Processing Unit)，通常简称为 CPU。它主要包括运算器、控制器、寄存器组和总线接口等。

运算器：算术逻辑部件 ALU。

控制器：指令寄存器、指令译码及机器周期编码器、定时及操作控制部件。

寄存器组：通用寄存器组、程序计数器及状态标志寄存器、指示器和变址寄存器、段寄存器组等。

总线接口部件：指令流字节队列缓冲器、存储器地址形成部件等。

对于高档微处理器来讲，运算器在执行部件中，控制器在控制部件里。

三、微型计算机

微型计算机是以微处理器(CPU)为中心，加上只读存储器(ROM)、读写存储器(RAM)以及输入/输出接口电路和系统总线缓冲器组成的。如图 1-2 所示。（注：系统总线缓冲器这里未画）

所谓位数是指微处理器可同时传送数据的数据总线的宽度。8086CPU 内部和外部数据总

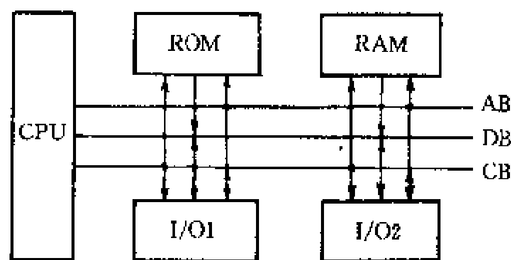


图 1-2 微型计算机结构框图

线为 16 位，所以 8086CPU 组成的计算机就是 16 位计算机。而 8088CPU 内部数据总线为 16 位，外部数据总线宽度为 8 位。由于它的结构与 8086 几乎是一样的，而在软件上完全是兼容的，能处理 16 位数据，因此称它为准 16 位微处理器。80386 为 32 位微处理，它传送数据的数据总线的宽度为 32 位。

1. 微处理器(CPU)

它是微型计算机的心脏，不仅把一般计算机中的控制器、运算器等集成在一个芯片上，而且它决定指令、指令系统。它能进行算术运算和逻辑运算，能够执行各种控制等。它的特性基本上反映了微型计算机的性能。各种不同类型的微处理器，都具有各自不同的一些特点，如指令系统、指令执行时间、控制能力、以及内部寄存器组、算术逻辑部件等硬件特性。这些硬件特性和指令系统在制造微处理器芯片时，是早已规定好了的，一般来讲用户是不能更改的，我们只好去熟悉它、掌握它、运用它。

2. 存储器

存储器是计算机极重要的组成部分。它是用来存储程序、原始数据、中间结果和最终结果的。有了它，计算机才能有记忆功能，才能把要计算和处理的数据以及程序存入计算机内，使计算机脱离人的直接干预，自动地工作。显然存储器容量越大，能记忆的信息就越多，计算机的功能就越强。由于存储器主要是和微处理器打交道，取指令、取操作数、存数等。而存储器存取速度是影响运算速度的主要因素，所以希望存储器容量要大，存取速度要快。

存储器现在所指的是半导体存储器，它可分为既可读又可写的存储器(RAM)，只能读出存储器(ROM)。RAM 又分为动态和静态存储器两种。ROM 分为掩模式 ROM，可编程 ROM (PROM)，可擦写的 ROM(EPROM)和电可擦写的 ROM(E^2PROM)。

(1)读写存储器(RAM)停电后信息会丢失

主要用来存放操作数据，处理信息时的中间结果以及最终结果。用一段存储单元作堆栈区等。

静态 RAM：在不停电条件下，存储单元所存储的信息不变的 RAM 叫静态 RAM。

动态 RAM：在不停电条件下，由于是用寄生电容来存储信息的，随时间变化因漏电而信息变化的叫动态 RAM。故每经过几个 ms 需要刷新一次，以保存信息。

(2)只读存储器(ROM)停电后信息不会丢失

主要用来存放各种固定的程序和数据，如高级语言的解释程序或编译程序、汇编程序、标准子程序、监控程序、用户自编控制程序等；也用来存储各种常用数据、表格等等。其程序和数据是制造厂或者是用户写入固化的。存储器芯片一般逻辑结构如图 1-3 所示

3. 输入输出

微处理器与其它数字计算机一样，利用外部设备与外界通信。常用的外部设备包括：键盘、显示器、行式打印机、CRT 显示设备、盒式磁带机、软硬磁盘、A/D 与 D/A 转换器等等。

当一个或几个外部设备与微处理器相连时，每个外部设备都必须有一个接口电路。这是因为：

(1)所使用外部设备的速度不尽相同，有快速的、慢速的、中速的之分，不可能与主机的工

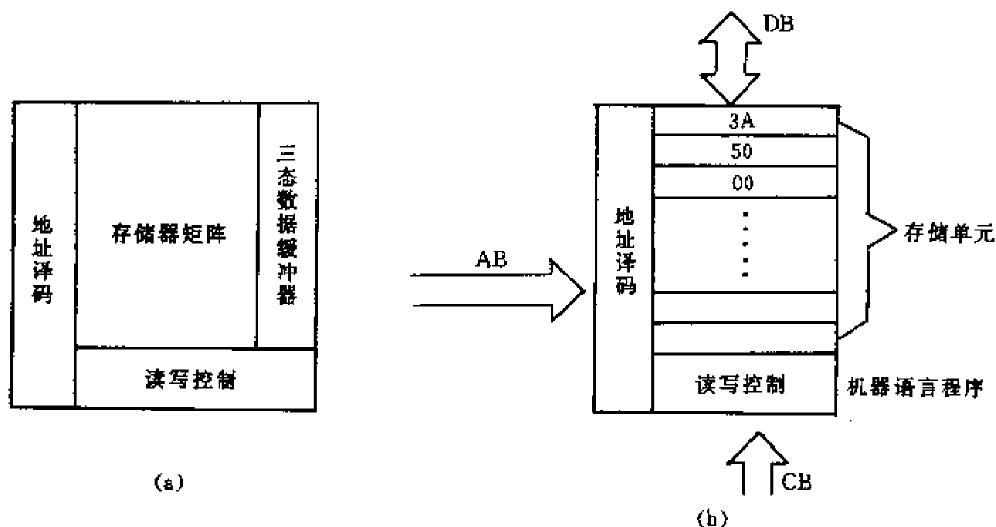


图 1-3　存储器芯片逻辑结构框图

作速度相匹配。

(2)CPU 输入和输出数据都是并行传输的。外部设备对数据格式的要求是各式各样的，例如有的 A/D 转换接口是 12 位的，有的要求串行传送等等。

(3)外部设备的结构各不相同，有电子式、机械式、机电式、电磁式等。使用的电路元件有 MOS 器件与 TTL 器件之分，因此，信号也要经过电平转换才能与 CPU 要求的信号相一致。CPU 与 I/O 之间的接口信号如图 1-4 所示。

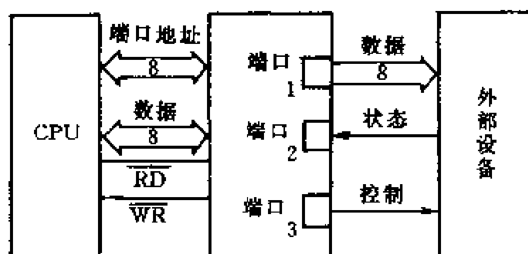


图 1-4　CPU 与外设之间接口

四、微型计算机系统

图 1-5 是微型机系统的示意图。由该图我们可以看出，微型机系统包含两大部分：硬件系统和软件系统。

硬件系统是在前述微型机的基础上配以必要的外部设备、外部存储器(如磁盘机，磁带机等)和电源设备等组成的。

软件系统包含了系统软件和应用软件。通常最基本的系统软件被固化在 ROM 存储器中，如监控程序、基本输入输出管理程序等。微型机系统的种类很多，从简单系统到复杂系统，从最小系统到最大系统，以满足社会各方面的不同要求。简单系统一般包含了微型计算机系统的基本配置：微型计算机、电源、磁盘驱动器、打印机及显示器。流行的个人计算机(PC 机)就属于这种系统。复杂的系统可以是多机系统、分布式系统等。本书将以单机系统为主要内容进行介

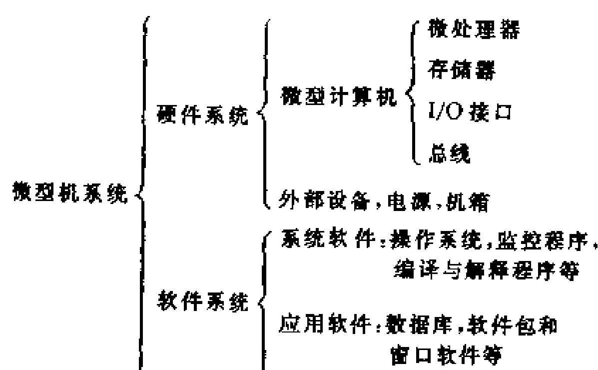


图 1-5 微型机系统示意图

绍。

五、微型计算机的开发系统

为研制微型计算机软件或对微型计算机的样机进行软硬件综合调试而专门制造的微型计算机系统，叫做微型计算机开发系统，它是设计和调试新微型计算机的强有力的工具。

功能完善的开发系统具有丰富的软件、多种外部设备，如外存储器、CRT、打印机等，一般都配有仿真器。这种多功能的开发系统可以进行仿真、跟踪和 EPROM 的读/写等。专用微型机开发系统是针对某种微处理器而研制的开发系统，例如 MOTOROLA 公司的 EXRCISER I 可以开发 M6800 及 M6809。通用的开发系统配有多种型号微处理器的仿真器，可以方便地开发多个系列的微型机系统，例如 TEKTRONIK 的 8500 系列开发系统，可以开发 8080/85、Z—80、M6800 等各种 8 位微型机和 8086/88 等 16 位微型计算机。多功能开发系统，不仅能开发多种微处理器，还能开发多机系统，具有仿真器和逻辑分析仪的功能，可以对硬件和软件进行综合调试。开发系统种类和功能不同，其价格也不相同。一般说来，价格高于普通微型计算机，用户应根据实际需要来选购。

六、微机结构的特点——总线技术

总线：计算机传送信息的一组信号线；

单向总线：只能向一个方向传送信息的总线；

双向总线：能向两个方向传送信息的总线。

微型计算机一般采用三总线，即按传输信息的性质分为三总线：地址总线、数据总线、控制状态总线。这样模块之间的连接就既方便，又易于扩展，便于构成多机系统。

微处理器内部总线

把内部寄存器组、累加器、算术逻辑单元和控制器部件集成在 $5\times5mm^2$ 芯片上，为了提高集成度采用了内部数据总线。

单机内总线

微型计算机内部系统板与插件板之间进行通信的总线，称为系统总线。如 IBM—PC 总线，AT 总线等。CPU 与外围芯片之间的总线称为内总线(局部总线)。

外总线：(多机之间)

微型计算机和其它设备或控制对象之间进行通信的总线叫外总线。有 IEEE－488 标准总线，EIA－RS232 异步通信总线，EIA－RS232&423 异步和同步通信标准总线，MULTIBUS 多总线等。

七、软件是微型计算机系统的重要组成部分

前面我们学习了构成微型计算机的硬件，但是，仅有这样的硬件，还只是具有了处理信息的基础。计算机真正能进行处理信息，还必须要有软件配合，软件即各种程序。

我们知道，计算机所以能脱离人的直接干预，自动地进行计算，是由于人把实现某个处理的步骤用命令的形式——一条条的指令序列预先输入到存储器中。执行时，机器把这些指令逐条地取出来，加以翻译和执行。

指令：计算机能够识别和执行的指挥计算机进行操作的命令的二进制编码（也叫做机器码）就是指令。如：取数、送数、相加、存数等等都是一种操作。一条指令对应一种基本操作。

指令系统：计算机所能执行的全部指令的集合。一台计算机能执行什么操作，能做多少种操作是由微处理机来决定的。换句话说，当微处理机设计好时，所规定的指令系统也就规定好了，这是计算机所固有的，不同的微处理器，指令系统是不一样的。

指令程序：能完成某种任务的计算机能够识别和执行的指令序列叫程序。

源程序：用户为解决自己的问题所编的程序，称为源程序。

目的程序：用计算机能够识别和执行的二进制编码（称为机器码）而编写的程序，叫做机器语言程序，又称为目的程序。

机器语言：机器码的集合。

由于计算机只认识二进制码，所以指令系统中所有指令，都必须以二进制编码形式来表示也称为机器语言。以 8086 为例，向存储器存数（字），机器码编码为 A3，06，00，30 四个字节（直接寻址）。从存储器取数，机器码编码为 8B，04，00，20 四个字节编码（变址寻址）等等。

用这些指令编码来编写的程序（机器码程序）是由一连串的 0 和 1 组成，没有明显特征，不好记忆，不易理解，出错查找困难，要求编程人员对机器码相当熟练。所以编写机器码程序是一件繁琐而困难的工作。机器语言优点是编程简短、运行迅速、可靠。为克服其缺点，人们就用一些助记符——通常是用指令功能英文词的缩写来代替操作码。如 8086/8088 中数的传送操作用助记符 MOV，加法用 ADD 等。这样一来，上面两条机器码编码可以写为

```
MOV [3000H],AX
MOV AX,[SI+2000H]
```

这样每条指令有明显的特征，易于理解和记忆，也不易出错。

汇编语言：用助记符、符号地址表示的指令叫汇编格式指令。

汇编格式指令：操作码　目的地址　源操作数或操作数地址。

操作码表示计算机执行什么操作，操作数表示参加操作数的本身或操作数所在的地址，其操作结果送到目的地址中。由汇编语言编写的源程序叫汇编语言源程序。由于计算机只认识二进制编码，对汇编语言它是不认识的，必须用一个汇编程序将汇编语言程序译成机器码（因汇编格式指令与机器码是一一对应的）。具有汇编能力的计算机，是由计算机进行汇编的；没有汇编能力的计算机，就得由人查指令表进行汇编。将汇编好了的机器码送到存储器中，以便计

算机自动执行。汇编语言的优点是好记、好理解、出错也好找、不需要熟记机器码。缺点是要求编程人员对计算机要有所了解。而每种机器其汇编语言不同，如 8086 传送为 MOV ，Z8000 则为 LD 等等，所以它不能通用。

为了克服汇编语言的不足，就出现了高级语言。如：BASIC、FORTRAN 、COBOL 、PASCAL、C 语言等。其优点是不用对计算机内部结构有所了解，不同机器能通用，可移植性好。缺点是同一个任务用高级语言编写的程序其执行时间比汇编语言要长，占用内存空间多。同样，用各种高级语言编写的程序计算机是不认识的，必须用语言处理程序翻译成机器码才能执行。编译程序是将高级语言程序全部翻译成目的（机器语言）程序，再由计算机执行的语言处理程序。而解释程序则是取一条高级语言语句，就解释一次，将其翻译成目的程序的语言处理程序。

为了管理机器和运行各种语言程序就出现了操作系统。

为了对信息进行处理，就出现了数据库和数据库管理系统程序等。

八、微型计算机硬件技术及其发展趋势

目前微型计算机基本上是沿着两个方向发展：一是生产性能更好的单片机及 4 位、8 位微型计算，主要是面向要求低成本的家电、传统工业改造及普及教育等。其特点是专用化、多功能、低价格、可靠性好；二是发展 16 位、32 位、64 位微型计算机，面向更加复杂的数据处理、OA、DA 科学计算等，其特点是大量采用最新技术成果，在 IC 技术、体系结构等方面向高性能、多功能的方向发展。

下面主要介绍一下微处理器所采用的硬件技术及其发展趋势。

1. 堆栈技术

后进先出原则。在内存开辟一区域，主要在用到子程序或中断时保存断点地址、保存现场内容等。这将在以后详细叙述。由于实际需要，计算机要与更多的外部设备打交道，顺序执行程序势必带来外设的串行工作，一外设执行完后另一个外设才开始投入工作，这样使 CPU 效率低（大部分时间花在等待外设工作完成上）。就用户使用来讲也是极不方便的，于是就出现了中断技术。

2. 中断技术

中断正在执行的程序，转到中断服务程序为申请中断的设备服务，使计算机同时为几个外设服务，大大提高了 CPU 的效率和微机系统的性能。这部分在第八章详述。

由于内存容量有限，大部分程序、数据是存放在外部存储器中的，如软盘、硬盘等。如果要执行某一程序，就必须将这部分程序调到内存中，CPU 才能一条条执行。CPU 与外设进行数据交换，一般是在输入/输出查询程序或中断程序控制下进行的。外设和内存之间的数据传送是在 CPU 的干预下进行的，传送速度慢，特别是大量数据传送就更显得慢了，于是就出现了 DMA 技术。

3. 直接存储器存取（DMA）技术

存储器和外设的 I/O 端口寄存器之间传送数据，采用不要 CPU 干预，直接在存储器和外设之间进行数据块传送，以提高传送数据的速度。为了提高计算机运行速度，在微处理器上也作了一定的改进。

4. 多寄存器结构

对微处理器来讲，内部寄存器越多，处理数据和地址就越容易，使用就越方便。CPU 处理

的中间结果，可以暂存在寄存器中，勿需送到存储器，使执行时间缩短，提高运行速度。

5. 流水线技术(重叠操作技术)

高档微处理器总的来说有总线接口单元和执行单元。总线接口是负责与总线相连的接口，实现 CPU 与存储器之间的信息传送，将指令或所需操作数取出送到指令队列中排队。而执行单元负责指令的执行，这样就把取指令部分与执行指令部分分开了，使取指和执行指令能够重叠进行，减少了等待取指时间，大大提高了 CPU 效率，提高了整机运行速度。另一方面又降低了对与之配备的存储器的存取速度的要求。

6. 高速缓冲存储器

在 32 位微处理器中，为了加快处理信息的速度，在 CPU 与常规主存之间放一个专用高速 RAM，它的速度比主存快一个数量级，不过容量小，只能存放一小组指令或数据集。CPU 要取指令或操作数时发出一个地址，它首先要看所需数据是否在高速缓存中，若在就立即送给 CPU，若不在就要做一次常规的存储器访问，将所需要的数或指令送给 CPU，并将其相邻的指令或数据放入高速缓存中。通常命中率高于 90%以上。

80486 已将高速缓冲存储器 82385 集成到了 486 微处理器中，486 执行速度是 25MHz 的 386 速度的 1.6 倍。

7. 虚拟存储器

被执行的实用程序和使用数据一般不允许大于主存 RAM 空间，若采用一种硬件和软件的综合技术，允许程序使用大于物理上的主存储器的容量，这是由操作系统启动存储器进行数据块的交换或覆盖技术来完成的。目前 80286、80386、80486 都具有虚拟存储器功能。

8. 多处理器系统技术

为了提高执行速度和改善系统操作能力，微型计算机大多采用了 DMA 控制器，通过窃取总线周期来完成 I/O 的操作。但 DMA 控制器的能力有限，且在 DMA 操作前后都要 CPU 干预，额外开销大。为了进一步提高运行速度和改善系统的操作能力，可采用多处理器系统(有两个或两个以上能同时译码和执行指令的部件，该系统就称为多处理器系统)。以 IBM－PC 机为例，它在最大组态时，可以组成多处理器系统，用 8087 数值数据处理器执行浮点运算，8089 的 I/O 处理器能够执行字符串处理、代码转换、字符搜集、位测试以及通常的 DMA 操作。这样一来 CPU 就能够处理更高级的任务了，微机运行效率大为提高，改善了系统的操作能力。

为了加快微处理器的研制，使用了微程序技术，平均 3～4 年就能研制出一个新的微处理器。

9. 微处理机普遍地采用了微程序控制技术

控制器由微程序控制技术组成，不使用组合逻辑控制，因为采用了组合逻辑控制器后，要想改变一下功能，组合逻辑表达式就得改变，从而硬件电路就得改变，这是做不到的。采用微程序设计就方便得多了。每个微命令控制完成一个微操作，由微命令组成微指令，由微指令编一段微程序，这段微程序用一些微操作完成一条机器指令的功能。若要改变一下功能，只需改变一下微程序就可以了。

10. 并行处理的哈佛(Harvard)结构

为了克服 MPU 数据总线宽度的限制，尤其是在单处理器情况下，为进一步提高微处理器的处理速度，采用高度并行处理技术－　Harvard 结构已成为引人注目的趋势。

哈佛结构的基本特性是：采用多个内部数据/地址总线；将数据和指令缓存的存取分开；使

MMU 和转换后援缓冲存储器(TLB)与 CPU 实现并行操作。该结构是一种非冯.诺依曼结构，其典型的代表产品是 Motorola 的 MC68030。

11. RISC 结构

所谓 RISC 结构就是精简指令集的微处理器结构。其指导思想是在微处理器芯片中，将那些不常用的由硬件实现的复杂指令改由软件来实现，而硬件只支持一些使用频度很高的基本指令。这种方法可以大大减少硬件的复杂程度，并能显著地减少处理器芯片中门的个数，可用砷化镓(GaAs)取代硅半导体材料制成微处理器。这种微处理器具有抗幅射、对温度不敏感、功耗低等优点。在恶劣环境下，这种微处理器性能良好，并且可以获得非常高的运算速度。但是，这种材料与硅相比，其加工技术难于掌握，技术还不成熟，芯片的集成度还远远满足不了传统的完善指令集计算机(CISC)的要求。所幸的是目前 RISC 技术已日臻成熟，为 GaAs 制造 RISC 结构的处理器芯片创造了良好的环境。

12. 整片集成技术(Wafer Scale Intergration)

目前高档微处理器已基本转向 CMOS VLS 工艺，集成度已突破百万晶体管大关。一个令人注目的动向是新一代的微处理器芯片已能将更多的功能部件集成在一起，并做在一个芯片上。目前在一个 MPU 的芯片上已实现了芯片上的存储管理、高速缓存、浮点协处理器部件、通信 I/O 接口、时钟定时器等功能。另外，单芯片多处理器并行处理技术也已由不少厂家研制出来。

从微型计算机系统角度来看，采用多机系统结构、增强图形处理能力、提高网络通信性能等方面都是当前微型计算机系统所追求的目标。

第三节　微型计算机的工作过程

微型计算机的工作就是运行程序，即逐条从存储器中取出程序中的指令并完成指令所指定的操作。

下面通过一个简单程序的执行过程，对微型计算机的工作过程做一个简单的介绍。

让我们看一下 0FH+08H 在计算机中是怎样实现的。其汇编语言程序如下：

ORG 00000H	机器码
MOV AL,0FH;	B0H
	0FH
ADD AL,08H;	04H
	08H
MOV [0009H],AL;	A2H
	09H
	00H
HLT	F4H

将机器码送到存储器中，如图 1-6 所示。

程序是从程序首地址 0000H 开始存放的。执行一条指令，首先要分析指令的操作码，然后

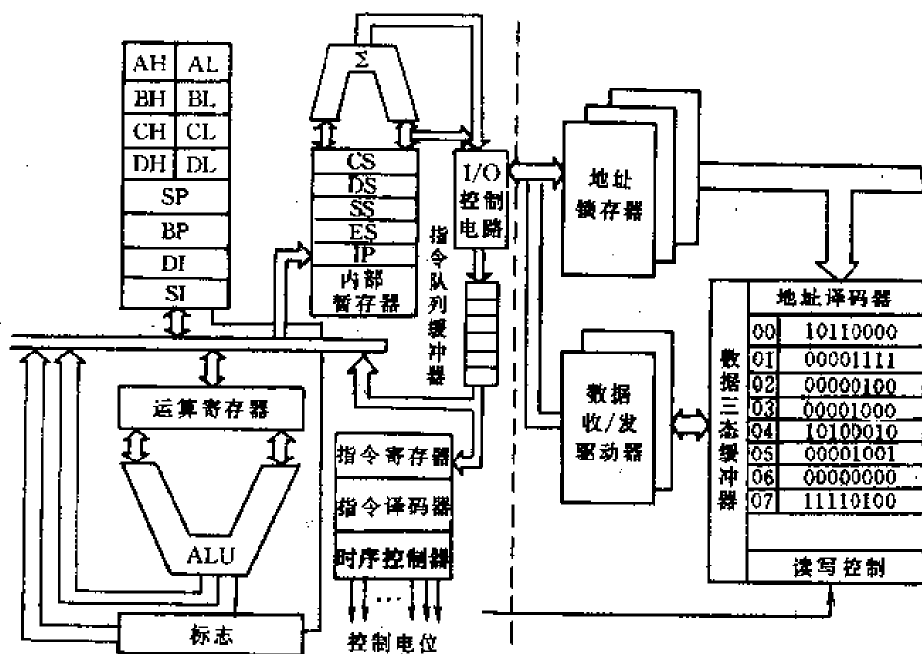


图 1-6 一个模型机执行流程图

再完成操作码规定的操作。其步骤如下：

(1) 程序计数器内容为 0000H。

(2) 把程序计数器内容 00H 和 CS 段值(设 0000H)相加得到地址为 00000H。

(3) 将 00000H 内容经 I/O 控制电路送外部 AD 总线上，地址由地址锁存器锁存，IP 自动加 1，准备取下一个机器码。

(4) 地址内容由地址锁存器输出送到存储器的地址译码器进行译码，找到 00H 单元。

(5) CPU 向存储器发读命令。

(6) 将 00H 单元内容 B0H 读出送到数据总线上，经数据收/发驱动器送 I/O 控制电路中。

(7) 读数被 I/O 控制电路送到指令队列缓冲器。

(8) 由于是操作码，故该数被送控制器中的指令寄存器，指令译码器进行译码，由时序操作控制部件发出相应于操作码的控制信息。

取操作数步骤如下：

(1) IP 将 01H 内容和 CS 内容相加，形成 00001H 送 I/O 控制电路，且 IP 自动加 1。

(2) 由 I/O 控制电路将 00001H 送 AD 线上，由地址锁存器将地址 00001H 锁存。

(3) 锁存器将地址信号送存储器的地址译码器进行译码，找到 01H 单元。

(4) CPU 发出读命令。

(5) 把操作数(01 单元内容)0FH 经数据三态缓冲器送至数据收/发驱动器，再送至 CPU 的 I/O 控制器电路。

(6) 由 I/O 控制电路将 0FH 送至指令队列缓冲器。

(7) 因 0FH 是操作数，故它将被送到操作码规定好的 AL 寄存器中。

上述步骤仅仅是完成一条指令 MOV AL，0FH，而其余指令也是需要分析操作码，然后完成操作码规定的操作的。这对每条指令来说都是相同的。但 IP 内容不同，所指的指令就不同。

第二章　计算机中的信息表示方法

计算机的基本功能是对数据进行加工，因此要加工的数据必须送入计算机中。人们习惯用十进制数，而计算机中却采用二进制，这是因为制作具有十个物理状态的器件很困难，而制作具有两个物理状态的器件却容易的多，省器件，且有成熟的逻辑处理工具，运算、处理也方便。所以在计算机中，所用的数字、字符、指令、状态都是用二进制数来表示。为了书写方便，计算机还采用其它进制数，如十六进制和八进制等。

第一节　数值及其编码

一、无符号数的表示及运算

（一）无符号数的表示方法

1．十进制计数的表示法

十进制计数法的特点是：

■以 10 为底，逢 10 进位。

■需要 10 个数字符号 0、1、2、…、9。

任何一个十进制数 N_D 可以表示为：

$$N_D=\sum_{i=-m}^{n-1} D_i\times 10^i \qquad (2.1.1)$$

其中，m 表示小数位的位数，n 表示整数位的位数，D_i 为十进制数字符号 0～9。

例如：

$$374.53D=3\times10^2+7\times10^1+4\times10^0+5\times10^{-1}+3\times10^{-2}$$

上式中后缀 D 表示十进制数(Decimal)，尾标识符 D 也可省略。

2．二进制计数的表示法

二进制计数法的特点是：

■以 2 为底，逢 2 进位。

■需要两个数字符号 0、1。

一个二进制数可以表示为如下形式：

$$N_B=\sum_{i=-m}^{n-1} B_i\times 2^i \qquad (2.1.2)$$

例如：

$$1101.1B=1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}$$

上式中后缀 B 表示二进制数(Binary)。

3．十六进制计数的表示法

十六进制计数法的特点是：

■以 16 为底，逢 16 进位。

■需要 16 个数字符号 0、1、2、...、9、A、B、C、D、E、F. 其中 A～F 依次表示 10～15。

一个十六进制数可表示为如下形式：

$$N_H=\sum_{i=-m}^{n-1} H_i\times 16^i \tag{2.1.3}$$

例如：E5AD.BFH $=14\times16^3+5\times16^2+10\times16^1+13\times16^0+11\times16^{-1}+15\times16^{-2}$

上式中后缀 H 表示十六进制数(Hexadecimal)。

一般来说，对于基数为 X 的任一数可以用多项式表示为：

$$N_X=\sum_{i=-m}^{n-1} K_i\times X^i \tag{2.1.4}$$

其中：X——基数，表示 X 进位制

i——位序号

K_i——第 i 位的系数，可以为 0、1、...、X−1 共 X 个数字符号中任一数字符号

m，n——m 为小数部分位数，n 为整数部分位数

X^i——第 i 位的权

（二）各种数制之间的转换

1. 任意进制数转换为十进制数

二进制、十六进制以至任意进制的数转换为十进制数的方法简单，可按式(2.1.1)、(2.1.2)、(2.1.3)、(2.1.4)公式展开求和即可。

2. 十进制整数转换为二进制数

由于整数部分与小数部分转换的规则不同，故整数部分与小数部分分别进行转换。

(1)十进制整数转换为二进制整数

任何一个十进制数转换为二进制数后，都可以表示成为式(2.1.2)的形式。问题的核心在于求出 n 及 B_i。

下面通过一个简单的例子分析一下转换的方法。例如，

已知 $13D=1101B=\underset{B_3}{1\times2^3}+\underset{B_2}{1\times2^2}+\underset{B_1}{0\times2^1}+\underset{B_0}{1\times2^0}$

上式也可以表示为：

$$\begin{aligned}13D=1101B &= (1\times2^2+1\times2)\times2+0\times2^1+1\times2^0\\ &= ((\underset{B_3}{1}\times2+\underset{B_2}{1})\times2+\underset{B_1}{0})\times2+\underset{B_0}{1}\end{aligned}$$

可见，要确定 13D 对应的二进制数，只需从右到左分别确定 B_0、B_1、B_2 和 B_3 即可。显然，从上式可以归纳出以下转换方法，即用 2 连续去除十进制数，直至商等于零为止。逆序排列余数便是与该十进制数相应的二进制数各位的系数值。过程如下：

```
2|13
 2|6      …1(商 6 余 1)—B0
  2|3     …0(商 3 余 0)—B1
   2|1    …1(商 1 余 1)—B2
    2|0   …1(商 0 余 1)—B3
```

∴ 13D = 1101B

用与此类似的方法也可以完成十进制数至十六进制数的转换，不同的是用16连续去除而已。

(2)十进制小数转换为二进制小数。

根据(2.1.2)式，

$$0.8125D = B_{-1}\times 2^{-1}+B_{-2}\times 2^{-2}+B_{-3}\times 2^{-3}+B_{-4}\times 2^{-4}$$
$$= 2^{-1}(B_{-1}+2^{-1}(B_{-2}+2^{-1}(B_{-3}+2^{-1}\times B_{-4})))$$

由上式可以看出，十进制小数转换为二进制小数的方法是，连续用2去乘十进制小数，直至乘积的小数部分等于"0"。顺序排列每次乘积的整数部分，便得到二进制小数各位的系数 $B_{-1},B_{-2},B_{-3},\cdots$。若乘积的小数部分永不为"0"，则根据精度的要求截取一定的位数即可。

0.8125D的转换过程如下：

0.8125D×2=1.625 得出 $B_{-1}=1$

0.625D×2=1.25 得出 $B_{-2}=1$

0.25D×2=0.5 得出 $B_{-3}=0$

0.50D×2=1.0 得出 $B_{-4}=1$

∴ 0.8125D = 0.1101B

可见，13.8125 = 1101.1101B

3. 二进制数与十六进制数之间的转换

因为 $2^4=16$，故二进制数转换为十六进制数只需以小数点为起点，向两端每4位(不足4位者补0)二进制用1位十六进制数表示即可。例如，十六进制数转换为二进制数时，将每1位十六进制数转换为相应的四位二进制数即可。

1101110.01011B=01101110.01011000B=6E.58H

二进制数书写冗长易错，因此一般用十六进制来表示数，比较简洁方便。

(三) 无符号二进制数的运算

1. 二进制数的算术运算

(1) 二进制加法

二进制加法运算规则为：0+0=0，

0+1=1，

1+0=1，

1+1=0(进位1)

(2) 二进制减法

二进制减法运算规则为：0−0=0，

1−1=0，

1−0=1，

0−1=1(有借位)

(3) 二进制乘法

二进制乘法运算规则为：0×0=1×0=0×1=0，

1×1=1

(4) 二进制除法

二进制除法是乘法的逆运算。

2. 二进制数的逻辑运算

(1) "与"运算(AND)

"与"运算又称为逻辑乘,可用符号"·"或"∧"表示。A、B两个逻辑变量进行"与"运算的规则如下:

A	B	A∧B
0	0	0
0	1	0
1	0	0
1	1	1

由上可知,只有当A、B两变量皆为"1"时,"与"的结果才为"1"。

(2) "或"运算(OR)

"或"运算又称为逻辑加,可用符号"+"或"∨"表示。A、B两个逻辑变量进行"或"运算的规则如下:

A	B	A∨B
0	0	0
0	1	1
1	0	1
1	1	1

由上可知,A、B两变量中,只要有1个为"1","或"运算的结果就是"1"。

(3) "非"运算(NOT)

变量A的"非"运算的结果用 $\overline{A}$ 表示,"非"运算规则如下:

A	$\overline{A}$
0	1
1	0

(4) "异或"运算(XOR)

"异或"运算用"$\forall$"表示,逻辑变量A,B进行"异或"运算的规则如下:

A	B	A$\forall$B
0	0	0
0	1	1
1	0	1
1	1	0

由上可知,A、B两变量只要不同,"异或"运算的结果就是1。

二、带符号数的表示及运算

(一) 带符号数的表示

日常生活中遇到的数除上述的无符号数外，还有带符号数。数的符号在计算机中也用二进制数表示，通常用二进制数的最高位表示数的符号。把一个数及其符号在机器中的表示加以数值化，这样的数称为机器数，而机器数所代表的数称为该机器数的真值。机器数可以用不同方法表示，常用的有原码、反码和补码表示法。

1. 原码

数 x 的原码记作$[x]_{原}$，如机器字长为 n，则原码的定义如下：

$$[x]_{原}=\begin{cases}x & 0\leqslant x\leqslant 2^{n-1}-1\\ 2^{n-1}+|x| & -(2^{n-1}-1)\leqslant x\leqslant 0\end{cases}$$

例如，当机器字长 $n=8$ 时，

$[+0]_{原}=00000000$，　　$[-0]_{原}=10000000$

$[+1]_{原}=00000001$，　　$[-1]_{原}=10000001$

$[+127]_{原}=01111111$，　　$[-127]_{原}=11111111$

当机器字长 n=16 时，

$[+0]_{原}=0000000000000000$，　　$[-0]_{原}=1000000000000000$

$[+1]_{原}=0000000000000001$，　　$[-1]_{原}=1000000000000001$

$[+32767]_{原}=0111111111111111$，　　$[-32767]_{原}=1111111111111111$

由此看出，在原码表示中，最高位为符号位，正数为 0，负数为 1。其余 n—1 位表示数的绝对值。

原码表示的整数范围是$-(2^{n-1}-1)\sim+(2^{n-1}-1)$。8 位二进制原码表示的整数范围是－127～＋127，16 位二进制原码表示整数的范围是－32767～＋32767。原码表示法简单直观，但不便于进行加减运算。

2. 反码

数 x 的反码记作$[x]_{反}$，如机器字长为 n，反码定义如下：

$$[x]_{反}=\begin{cases}x & 0\leqslant x\leqslant 2^{n-1}-1\\ (2^{n}-1)-|x| & -(2^{n-1}-1)\leqslant x\leqslant 0\end{cases}$$

例如，当机器字长 n＝8 时，

$[+0]_{反}=00000000$，　　$[-0]_{反}=10000000$

$[+1]_{反}=00000001$，　　$[-1]_{反}=11111110$

$[+127]_{反}=01111111$，　　$[-127]_{反}=10000000$

从反码表示法中可见，最高位仍为符号位，正数为 0，负数为 1。反码表示整数的范围是$-(2^{n-1}-1)\sim+(2^{n-1}-1)$。8 位二进制数反码表示整数的范围是－127～＋127，16 位二进制数反码表示整数的范围是－32767～＋32767，与原码相同。

从上例中还可以看出，正数的反码与原码相同，负数的反码只需将其对应的正数按位求反即可得到。

3. 补码

根据同余的概念：$a+NK=a\ (\mathrm{mod}\ K)$

其中 K 是模，N 为任意整数。就是说，在模的意义下，数 a 与该数本身加上其模的任意整

数倍之和相等。

在数 a 的无数个 a+NK 同余数中，我们感兴趣的是 N 为 1 的同余数，即补数：

$$[a]_{补数} = a + K\ (mod\ K) = \begin{cases} a, & 0 \leqslant a < K \\ K - |a| & -K \leqslant a < 0 \end{cases}$$

由上式确定的两种条件下数 a 的补数，正是补码的定义和补码编码规则的基础。在计算机运算过程中，数据的位数，即字长总是有限的。这里假设字长为 n，两数相加求和时，如果 n 位的最高位产生了进位，进位位就会丢掉。这正是在模的意义下相加的概念。相加时丢掉的进位即等于模。所以，当 n 位表示整数时(1 位为符号位，n－1 位为数值位)，它的模为 2n，即：

$$\underbrace{100\ldots\ldots0}_{n 个 0}$$

我们把 x 的补码记为$[x]_补$，补码的定义为：

$$[x]_补 = \begin{cases} x & 当\ 0 \leqslant x < 2^{n-1} \\ & \qquad\qquad (mod\ 2^n) \\ 2^n + x & 当\ -2^{n-1} \leqslant x < 0 \end{cases}$$

从定义式可见，正数的补码与其原码相同，只有负数才有求补的问题。所以，严格地说，“补码表示法”应称为负数的补码表示法。一个二进制数，以 2^n 为模，它的补码叫做 2 的补码。由上述推导，我们可得出以下结果。

数 x 的补码记做$[x]_补$，当机器字长为 n 时，补码定义如下：

$$[x]_补 = \begin{cases} x & 当\ 0 \leqslant x < 2^{n-1} - 1 \\ 2^n - |x| & 当\ -2^{n-1} \leqslant x < 0 \end{cases}$$

例如，当机器字长 n=8 时，

$[+0]_补=00000000$，　　$[-0]_补=00000000$

$[+1]_补=00000001$，　　$[-1]_补=2^8-|-1|=11111111$

$[+127]_补=01111111$，　　$[-127]_补=2^8-|-127|=10000001$

补码表示法中，最高位仍为符号位，正数为 0，负数为 1。

补码表示的整数范围为$-2^{n-1}\sim+(2^{n-1}-1)$。8 位二进制补码表示的整数范围为－128～＋127，16 位二进制补码表示的整数范围是－32768～＋32767。

从上例中还可以看出，正数的补码与它的原码、反码均相同，负数的补码等于它的反码加末位 1，也就是说，负数的补码等于其对应正数的补码按位求反(包括符号位)再加末位 1。如上例中，求－127 的补码过程可以简化如下：

$$\overline{[-127]} = \overline{[+127]_补} + 1 = \overline{01111111} + 1 = 10000001$$

8 位二进制数的原码，反码和补码列表如表 2-1，以供参考。

表 2-1 原码、反码和补码表

二进制数	无符号数	带符号数		
		原码	补码	反码
00000000	0	+0	+0	+0
00000001	1	+1	+1	+1
00000010	2	+2	+2	+2
…	…	…	…	…
01111110	126	+126	+126	+126
01111111	127	+127	+127	+127
10000000	128	−0	−128	−127
10000001	129	−1	−127	−126
…	…	…	…	…
11111101	253	−125	−3	−2
11111110	254	−126	−2	−1
11111111	255	−127	−1	−0

(二) 真值与补码之间的转换

1. 真值转换为补码

根据补码的定义便可以完成真值到补码的转换。

2. 补码转换为真值

(1) 正数补码转换为真值

由于正数的补码是其本身，因此，正数补码的真值 $x=[x]_{补}(0\leqslant x\leqslant 2^{n-1}-1)$。

(2)负数补码转换为真值

负数补码和与对应的正数补码之间存在如下关系：

$$[x]_{补}\xrightarrow{\text{求补运算}}[-x]_{补}\xrightarrow{\text{求补运算}}[x]_{补}$$

其中，x 是带符号数，正负皆可。求补运算是将一个二进制数按位求反加末位 1 的运算。对此关系此处不加证明，只用实例说明。

设：$x=+1$，则 $-x=-1$。

已知：$[x]_{补}=[+1]=00000001$

对$[x]_{补}$做求补运算的过程如下：

$$\overline{[x]_{补}}+1=\overline{[+1]}_{补}+1=\overline{0000001}+1=11111111=[-1]_{补}=[-x]_{补}$$

由这个例子可以看出，当 x 为正数时，对其补码进行求补运算，结果是 $-x$ 的补码。

若设：$x=-1$，则 $-x=+1$。

已知：$[x]_{补}=[-1]_{补}=11111111$。

对$[x]_{补}$做求补运算的过程如下：

$$\overline{[x]}_{补}+1=\overline{[-1]}_{补}+1=\overline{11111111}+1=00000001=[+1]_{补}=[-x]_{补}$$

由此例可以看出，当 x 为负数时，对其补码进行求补运算，结果是 $-x$ 的补码。

上述两例验证了$[x]_{补}$与$[-x]_{补}$之间的关系。显然，对负数补码进行求补运算的结果是该

负数对应的正数的补码，也就是该负数的绝对值。因此，负数补码转换为真值的办法如下：将负数补码按位求反末位加 1(即求补运算)即可得到该负数补码对应的真值的绝对值。也就是说，对负数而言，$|x|=\overline{[x]_{补}}+1$。

例 2.1 求下列数的补码

(1)设 $x=+127$，求$[x]_{补}$。

应用十进制转换为二进制数的原则，可以得出 $x=01111111B$。

故$[x]_{补}=[+127]_{补}=01111111$。

(2)设 $x=-127$，求$[x]_{补}$。

因为对$[x]_{补}$进行求补运算便可得到$[-x]_{补}$。因此，

$$[x]_{补}=[-127]_{补}=\overline{[+127]_{补}}+1=\overline{01111111}+1=10000001。$$

例 2.2 求以下补码的真值。

(1)设$[x]_{补}=01111110$，求 x。

因此该补码的最高位为“0”，即符号位为“0”，该补码对应的真值是正数。

则 $x=[x]_{补}=01111110=+126D$。

(2)设$[x]_{补}=10000010$，求 x。

因为该补码的最高位为“1”，即符号位为“1”，该补码对应的真值是负数，其绝对值为：

$|x|=\overline{[x]_{补}}+1=\overline{10000010}+1=01111101+1=01111110=+126$，则 $x=-126$。

(三) 补码的运算

1. 补码加法的规则是 $[x+y]_{补}=[x]_{补}+[y]_{补}$

其中，x，y 为正负数皆可，下面用四个例子来验证这个公式的正确性。

已知：$[+51]_{补}=00110011$ $[+66]_{补}=01000010$

$[-51]_{补}=11001101$ $[-66]_{补}=10111110$

则：

十进制加法	二进制（补码）加法
① + 66	01000010 = $[+66]_{补}$
+) + 51	+) 00110011 = $[+51]_{补}$
+117	01110101 = $[+117]_{补}$
② + 66	01000010 = $[+66]_{补}$
+) − 51	+) 11001101 = $[-51]_{补}$
+ 15	1 00001111 = $[+15]_{补}$
③ − 66	10111110 = $[-66]_{补}$
+) + 51	+) 00110011 = $[+51]_{补}$
− 15	11110001 = $[-15]_{补}$

④ −66	10111110 = $[-66]_{补}$
+) −51	+) 11001101 = $[-51]_{补}$
−117	1 10001011 = $[-117]_{补}$

2. 补码减法

补码的减法规则是：

$$[x-y]_{补}=[x]_{补}+[-y]_{补}$$

下面仍用四个例子对此规则的正确性加以验证。

十进制减加法	二进制（补码）减法
① +66	01000010 = $[+66]_{补}$
−) +51	+) 11001101 = $[-51]_{补}$
+15	1 00001111 = $[+15]_{补}$
② +66	01000010 = $[+66]_{补}$
−) −51	+) 00110011 = $[+51]_{补}$
+117	01110101 = $[+117]_{补}$
③ +51	00110011 = $[+55]_{补}$
−) +66	+) 10111110 = $[-66]_{补}$
−15	11110001 = $[-15]_{补}$
④ −51	11001101 = $[-51]_{补}$
−) −66	+) 01000010 = $[+66]_{补}$
+15	1 00001111 = $[+15]_{补}$

三、定点数和浮点数

在计算机中，用二进制数表示实数的方法有两种，即定点数和浮点数。

1. 定点数

所谓定点数，即小数点在数中的位置是固定不变的。以定点法表示的实数叫定点数。通常，定点数表示有两种方法。

方法 1：规定小数点固定在最高数值位之前，机器中能表示的所有数都是小数。n 位数值部分所能表示的数 N 的范围，用原码表示是：

$$1-2^{-n}\geqslant N\geqslant-(1-2^{-n})$$

它能表示的数的最大绝对值为 $1-2^{-n}$。最小绝对值为 2^{-n}。

方法 2：规定小数点固定在最低数值位之后，机器中能表示的所有数都是整数。n 位数值部分所能表示的数 N 的范围用原码表示是：

$$2^{n}-1\geqslant N\geqslant-(2^{n}-1)$$

它能表示的数的最大绝对值为 2^{n-1}，最小绝对值为 1。

图 2-1 给出定点数的两种表示法。

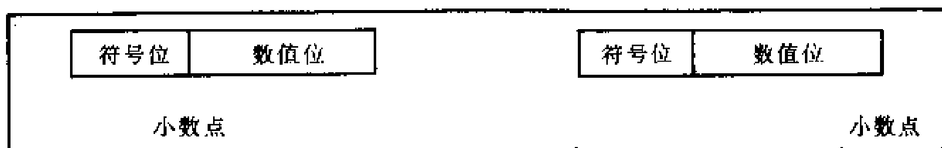


图 2-1　定点数的两种表示法

因为实际数值很少有都是小数或都是整数的，所以定点表示法要求程序员做的一件重要工作是为要计算的问题选择“比例因子”。所有原始数据都要用比例因子化成小数或整数，计算结果又要按比例因子恢复为实际值。在计算过程中，中间结果若超过最大绝对值，机器便产生溢出，叫做“上溢”。这时必须重新调整比例因子。中间结果若超过最小绝对值，计算机只能把它当作 0 处理，叫做“下溢”。这时也必须重新调整比例因子。对于复杂计算，计算中间需多次调整比例因子。

2. 浮点数

任意一个二进制数 N 总可以写成下面的形式：

$$N = \pm d \cdot 2^{\pm p}$$

其中：d 称为尾数，是二进制纯小数，指明数的全部有效数字。前面的符号称作数符，表示数的符号，用尾数的最高位表示。0 表示正，1 表示负；p 称为阶数，它的符号位称作阶符，用阶码最高位表示。阶符为正时，用 0 表示，阶符为负时，用 1 表示。由此可知，将尾部 d 的小数点向右（对+p）或向左（对−p）移动 p 位，即得数值 N。所以阶符和阶码指明小数点的位置。小数点随着 p 的符号和大小而浮动。这种数称为浮点数。浮点数的编码格式如图2-2所示。

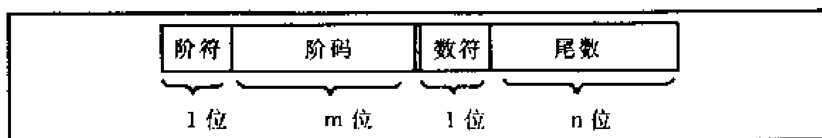


图 2-2　浮点数的编码格式

设阶码的位数为 m 位，尾数的位数为 n 位，则浮点数的取值范围为：

$$2^{-n} \cdot 2^{-(2^m-1)} \leqslant |N| \leqslant (1-2^{-n}) \cdot 2^{(2^m-1)}$$

浮点数能表示的数值范围大，是它的主要可取之处。

如果尾数的绝对值小于 1 且大于等于 0.5，即采用原码编码的正数或负数和采用补码编码的正数，其尾数的最高位数字为 1；采用补码编码的负数，其尾数的最高位数字为 0，则该浮点二进制数被称为规格化浮点数。浮点运算后，经常要把结果规格化。规格化的操作是尾数每右移 1 位（相当于小数点左移 1 位），阶码加 1；尾数每左移 1 位，阶码减 1。

数的加减运算要求小数点对齐。对于浮点表示的数而言，就是阶码（包括阶符）相等。使阶码相等的操作称为对阶。一个浮点数的阶码的改变，必须伴随着尾数的移位，才不改变数的值。即阶码加 1，尾数必须右移 1 位；阶码减 1，则要求尾数左移 1 位。两个规格化的浮点数相加或相减之前必须对阶。对阶的规则是：将两个数中阶码小的数的尾数右移，阶码增大，直到与另一个数的阶码相等为止。这样操作是合理的，因为尾数右移，只可能丢失最低有效

3后

位，造成误差较小。

可见，完成浮点数加法或减法运算，需要进行对阶、求和、规格化、舍入、判溢出等步骤，下面举例说明。

例如，设 $x=2^{010}\times0.11011011$，$y=2^{100}\times(-0.10101100)$，求 $x+y$。

解：假设两数均以补码表示，且采用双符号位，则它们的浮点表示分别为：

$[x]_{浮}=00\ 010,00.11011011$

$[y]_{浮}=00\ 100,11.01010000$

（1）求阶差及对阶

$\triangle_E=E_x-E_y=[E_x]_补+[-E_y]_补=00\ 010+11\ 100=11\ 110$

即 $\triangle_E$ 为 -2，x 的阶码小，应使 M_x 右移 2 位，E_x 加 2，

$[x]_{浮}=00\ 100,\ 00.00110110(11)$

其中(11)表示 M_x 右移两位后移出的最低两位数。

（2）尾数求和

$$
\begin{array}{r}
00.00110110(11)\\
+\ 11.01010100\\
\hline
11.10001010(11)
\end{array}
$$

（3）规格化处理

尾数运算结果的符号位与最高数值为同值，应执行左规处理，结果为 11.00010101(10)，阶码为 00011。

（4）舍入处理

采用 0 舍 1 入法处理，则有

$$
\begin{array}{r}
11.00010101\\
+1\\
\hline
11.00010110
\end{array}
$$

（5）判溢出

阶码符号位为 00，不溢出，故得最终结果为：

$x+y=2^{011}\times(-0.11101010)$

符点数的乘除法方法与加减法类似，这里就不详细说明了。

四、二进制编码的十进制数(BCD 码)

尽管计算机采用的二进制数的表示法及运算规则简单，但书写冗长，不直观。有的场合需要计算机的输入输出仍采用人们习惯的十进制数。当然，一位十进制数在计算机中需用四位二进制编码表示。这种编码有多种形式，其中 8421BCD(Binary—Coded Decimal)码比较常用。四位二进制有 16 种组合状态，当用来表示十进制数时，舍去高 6 种组合状态。BCD 码有两种形式，即压缩 BCD 码和非压缩 BCD 码。

1. 压缩 BCD 码

压缩 BCD 码的每一位用 4 位二进制表示，一个字节表示两位十进制数。例如 10010110B 十进制数 96D

2. 非压缩 BCD 码

非压缩 BCD 码用 1 个字节表示一位十进制数，只用低 4 位的 0000～1001 表示 0～ 9 。例如，00001000B 表示十进制数 8，或者是 0～9 的 ASCII 码。

两种 BCD 码的部分编码如表 2-2 所示。

表 2-2　BCD 码表

十进的制表	压缩 BCD 码	非压缩 BCD 码
0	00000000	0000000
1	00000001	0000001
2	00000010	00000010
⋮	⋮	⋮
9	00001001	00001001
10	00010000	00000001　00000000
11	00010001	00000001　00000001
⋮	⋮	⋮

第二节 计算机中常用的字符编码

一、字符编码(ASCII 码)

现代计算机使用各种程序设计语言，任何语言都是由字母、数字和符号组成的。要输入程序，计算机必须接受由字母、数字和符号组成的信息；用户在机器上操作时，总要输入许多监控程序或操作系统所能识别的各种命令，命令也是由字母，数字和符号组成的。计算机输出也是这样，它把人们可以识别的字母、数字和符号打印出来或显示在屏幕上。

在计算机内，任何信息都是用代码表示的。字母、数字和符号(以后简称为字符)也是用代码表示的。一般情况下，计算机依靠输入设备把要输入的字符转换成为一定格式的代码，然后才能接收进来。输出则是相反过程。为了在输出设备输出字符，计算机要把相应字符的编码送到外部输出设备。

目前国际上使用的字符编码系统有许多种，在微型计算机中普遍采用的是美国标准信息交换代码，即 ASCII 码(American Standard Code for Information－Interchange)。ASCII 码采用 7 位二进制代码来对字符进行编码。它包括 32 个通用控制符号，10 个阿拉伯数字，52 个英文大，小写字母，34 个专用符号，共 128 个。例如阿拉伯数字 0～9 的 ASCII 码分别为 30H～39H，英文大写字母 A、B、...、Z 的 ASCII 码是从 41H 开始依次往下编码，并非所有的 ASCII 字符都能打印；有些字符用来控制终端退格、换行和回车等。ASCII 代码还包括几个其它的字符，例如文件结束(EOF)、传送结束(EOT)、用作存储和传送数据的标志等。

通常，7 位 ASCII 代码的最高位为逻辑“0”。因此，字符在计算机内部按 8 位一组存储，正好占一个字节。在存储、处理和传送信息时，最高位常用作奇偶校验位，用来检验代码在存储和传送过程中是否发生错误。偶校验时，每个代码的二进制形式中应有偶数个 1。例如传送字母“G”，其 ASCII 码的二进制形式为 1000111，因有 4 个 1，故偶校验位为 0。8 位代码将是 01000111 。奇校验每个代码中应有奇数个 1。若用奇校验传送字符“G”，则 8 位代码为 11000111。奇偶校验只具有发现代码在存储和传送过程中奇数个位发生错误的能力。但它简

单可行，所以在计算机中，被广泛地用于信息的存储和传送中。

二、汉字编码(国标码)

1. 汉字的输入编码

为了能直接使用西文标准键盘把汉字输入到计算机，就必须为汉字设计相应的输入编码方法，当前采用的方法主要有以下三类：

(1) 数字编码

常用的是国标区位码，用数字串代表一个汉字输入，区位码是将国家标准局公布的 676 3 个两级汉字分为 94 个区，每个区分 94 位。实际上把汉字表示成二维数组，每个汉字在数组中的下标就是区位码，区码和位码各两位十进制数字，因此输入一个汉字需按键四次。例如："中"字位于第 54 区 48 位，区位码为 5448。

数字编码输入的优点是无重码，且输入码与内部编码的转换比较方便，缺点是代码难以记忆。

(2) 拼音码

拼音码是以汉语拼音为基础的输入方法，凡掌握汉语拼音的人，不需训练和记忆，即可使用，但汉字同音字太多，输入重码率很高，因此按拼音输入后还必须进行同音字选择，影响了输入速度。

(3) 字形编码

字形编码是用汉字的形状来进行的编码，汉字总数虽多但都是由一笔一划组成的，全部汉字的部件和笔划是有限的。因此把汉字的笔划部件用字母或数字进行编码，按笔划的顺序依次输入，就能表示一个汉字。例如五笔字形编码就是最有影响的一种字形编码方法。

除了上述三种编码方法之外，为了加快输入速度，在上述方法基础上，发展了词组输入、联想输入等多种快速输入方法，但是这些方法都利用了键盘进行"手动"输入。理想的输入方式是利用语音或图象识别技术"自动"将拼音或文本输入到计算机内，使计算机能认识汉字，听懂汉语，并将其自动转换为机内代码表示，目前这种理想已经成为现实。

2. 汉字内码

汉字内码是用于汉字信息的存储、交换、检索等操作的机内代码，一般采用两个字节的二进制形式表示，英文字符的机内代码是七位的 ASCII 码，当用一个字节表示时，最高位为"0"。为了与英文字符能相互区别，汉字机内代码中两个字节的最高位均规定为"1"。例如汉字操作系统 CCDOS 中使用的汉字内码就是一种最高位为"1"的两字节内码。

有些系统中字节的最高位用于奇偶校验位，这种情况下用三个字节表示汉字内码。

3. 汉字字模码

字模码是用点阵表示的汉字字形代码，它是汉字的输出形式。

根据汉字输出的要求不同，点阵的多少也不同。简易型汉字为 16×16 点阵，提高型汉字为 24×24 点阵、32×32 点阵，甚至更高，因此字模点阵的信息量是很大的，所占存储空间也很大。以 16×16 点阵为例，每个汉字要占用 32 个字节，国标两级汉字要占用 256K 字节。因此字模只能用来构成汉字库，而不能用于机内存储。字库中存储了每个汉字的点阵代码，当显示输出或打印输出时检索字库，输出字模点阵，得到字形，图 2-3 示出了"英"字的点阵及编码。

注意，汉字的输入编码、汉字内码、字模码是计算机中用于输入、内部处理、输出三种不同

的编码，不要混为一谈。

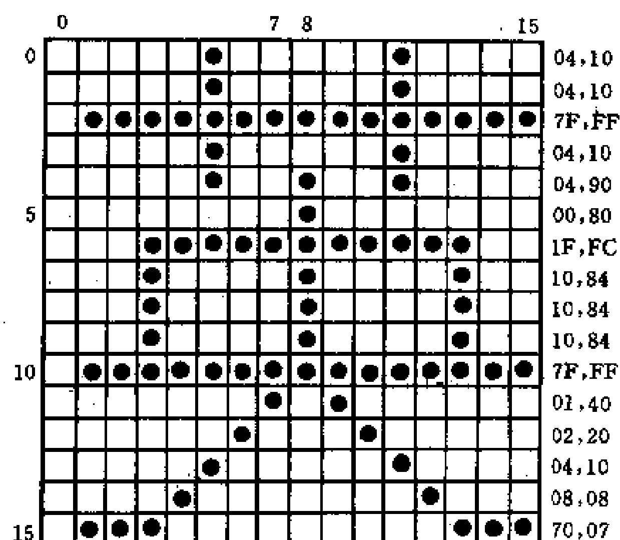


图 2-3　汉字的字模点阵及编码

思 考 题 与 习 题

1. 将下列十进制数分别转换成二进制、八进制、十六进制数和 BCD 数。

(1) 113.8125　　(2) 351.518　　(3) 957.84375　　(4) 538.375

2. 将下列二进制数分别转换成十进制、八进制、十六进制数和 BCD 数。

(1) 10110110.0011　　(2) 101.101101

(3) 1001.01011　　(4) 10011001.101

3. 将下列十六进制数分别转换成二进制、八进制、十进制和 BCD 数。

(1) 5D.BA　　(2) 12.C1　　(3) 93D.5D　　(4) E4B.7C

4. 完成下列二进制数的运算：

(1) 101+1.01　　(2) 1010.001−10.1

(3) −1011.01101−1.1001　　(4) 10.111×10.01

(5) 110011/11　　(6) (−101.01)/(−0.1)

5. 完成下列十六制数的运算：

(1) 11.A+8D2.8F　　(2) 5D.16+A4.95

(3) E27.5C−5B.E2　　(4) 4C.1D−E2D.F

6. 完成下列 BCD 数的运算，并按二 — 十进制调整的规律进行调整。

(1) 00100101+00110111

(2) 001101101000+011110010100

(3) 01100001−00100110

(4) 100001010111−000101101001

7. 完成下列逻辑运算：

(1) 10110101 ∨ 11110000　　(2) 11010001 ∧ 10101011

(3) 10101011⊕00011100

8. 写出下列字符的 ASCII 码(查 ASCII 码表)

(1)T　　(2) 2　　(3) P　　(4) >

(5)DLE　　(6)DEL　　(7)ACK　　(8) ETB

9. 写出下列十进制数的原码、反码和补码表示(用8位二进制数表示,最高位为符号位)。

A. 阶符与数符相同为规格化数。

B. 阶符与数符相异为规格化数。

C. 数符与尾数小数点后第一位数字相异为规格化数。

D. 数符与尾数小数点后第一位数字相同为规格化数。

11. 若X=0.1011，Y=-0.0101 求$[-X]_{补}$码，$[1/2\ X]_{补}$码，$[1/4\ X]_{补}$码，$[-Y]_{补}$码，$[1/2\ Y]_{补}$码，$[1/4\ Y]_{补}$码。

12. 设机器字长为32位，定点表示时，数符1位，尾数31位；浮点表示时，阶符1位，阶码5位，数 符1位，尾数25位。

(1) 定点原码整数表示时，最大正数为多少？最小负数为多少？

(2) 定点原码小数表示时，最大正数为多少？最小负数为多少？

(3) 浮点原码整数表示时，最大浮点数为多少？最小浮点数为多少？

13. 设用补码表示的二进制浮点数，阶符1位，阶码2位，数符1位，尾数12位。

(1) 最大正数是多少？

(2) 最小正数是多少？

(3) 最大负数是多少？

(4) 最小负数是多少？

14. 某浮点数基值为2(即阶码的底)，阶符1位，阶码3位，数符1位，尾数7位，阶码和尾数均为 补码表示，且尾数采用规格化数表示. 它所能表示的最大正数真值是多少？非零最小正 数真值是多少？绝对值最大的负数真值是多少？绝对值最小的负数真值是多少？

15. 已知X和Y，采用单符号位求$[X+Y]_{补}$码，结果是否溢出？

(1) X=0.11001，　　Y=-0.10010

(2) X=0.11001，　　Y=-0.10111

16. 已知X和Y，采用单符号位求$[X+Y]_{补}$码，结果是否溢出？

(1) X=0.11011，　　Y=-0.00111

(2) X=-0.01111，　　Y=0.00101

17. 用补码运算方法求 X+Y=?

(1) X=0.1001，　　Y=0.1100

(2) X=-0.0100，　　Y=0.1001

18. 用补码运算方法求 X-Y=?

(1) X=-0.0100，　　Y=0.1001

(2) X=-0.1011，　　Y=-0.1010

19. 已知X=0.1010，Y=-0.0110. 用补码一位乘法计算X×Y=?

20. 已知X=0.10110，Y=0.111113. 用补码加减交替法计算X/Y=?

21. 设有两个二进制数：$X=-0.875\times2^4$，$Y=0.625\times2^2$。

(1) 将X，Y的尾数转换为二进制补码形式。

(2) 设阶符1位，阶码2位，数符1位，尾数3位，用补码运算规则求X-Y的二进制浮点规格化 结果。

22. 设$[X]_{补}=X_0X_1X_2...X_n$，

求证：$X=-X_0+\sum X_i\times2^{-i}$。

23. 求证：$-[Y]_{补}=+[-Y]_{补}$。

24. 求证：$[X]_{补}=[X]_{反}+2^{-n}$。

25. 数的定点表示和浮点表示各有什么特点？

第三章　80X86 系列微型计算机的体系结构

微型计算机是由具有不同功能的一些部件组成的，微处理器(CPU)是微型计算机的心脏。它决定了微型计算机的结构，要构成一个微型计算机，必须首先了解 CPU 的结构和电气性能。

自 1971 年 INTEL4004 问世以来，微处理器的发展速度惊人。1978 年该公司推出了 8086 微处理器。8086 微处理器显示出强大的生命力，其性能已十倍于 8 位微处理器。1982 年推出更高性能的 16 位微处理器 80286，到 85 年又推出了 32 位微处理器 80386。而 1989 年又开发出更新结构的 80486，一直到 93 年推出了全新的 P5(Pentium)。短短十几年时间里，微处理器芯片集成度提高了 100 多倍，主频提高十多倍，运算速度达 100MIPS，其性能已达中、小型机水平，并且具有 RISC CPU 水平和某些智能特征。这些微处理器构成 80X86 系列。下面介绍 8086/8088CPU，其它 CPU 在第十章作较详细的介绍。

第一节 8086/8088 CPU

8086 是 INTEL 系列的 16 位微处理器，它是采用 HMOS 工艺技术制造的，内部包含约 29000 个晶体管。

8086 有 16 根数据线和 20 根地址线，因为可用 20 位地址，所以可寻址的地址空间达 2^{20} 即 1M。

8086 工作时，只要单一的 5V 电源和单相时钟，时钟频率为 5MHz。后来，INTEL 公司推出的 8086－1 型微处理器时钟频率高达 10MHz，8086－2 型微处理器时钟频率达 8MHz。

几乎在推出 8086 微处理器的同时，INTEL 公司还推出了一种准 16 位微处理器 8088。推出 8088 的主要目的是为了与当时已有的一整套 INTEL 外围接口芯片直接兼容使用。8088 的内部寄存器，内部运算部件以及内部操作都是按 16 位设计的，但对外的数据总线只有 8 条。在本章的讲述过程中，我们对 8088 也将作必要的说明。

一、8086/8088 CPU 的编程结构

所谓编程结构，就是指从程序员和使用者看到的结构。这种结构与 CPU 的内部物理结构和实际布局是有区别的，在 8086CPU 的编程结构(图 3-1)中可以看到，从功能上可分为总线接口部件 BIU(BUS INTERFACE UNIT)和执行部件 EU(EXECUTION UNIT)。

1. 功能结构

执行单元 EU 由 8 个 16 位的通用寄存器、1 个 16 位的标志寄存器、16 位的算术逻辑单元 ALU 及 EU 控制电路组成，总线接口单元 BIU 包括 4 个 16 位的段寄存器、1 个 16 位的指令指针寄存器、一个与 EU 通讯的内部寄存器、先入先出的指令队列、总线控制逻辑及计算 20 位物理地址的地址加法器 $\sum$。8088 的指令队列长为 4 个字节，而 8086 为 6 个字节。

EU 的功能是执行指令。EU 从指令队列取出指令代码，将其译码，发出相应的控制信息。控制数据在 ALU 中进行运算，运算结果的特征保留在标志寄存器 FLAGS 中。BIU 的功能是

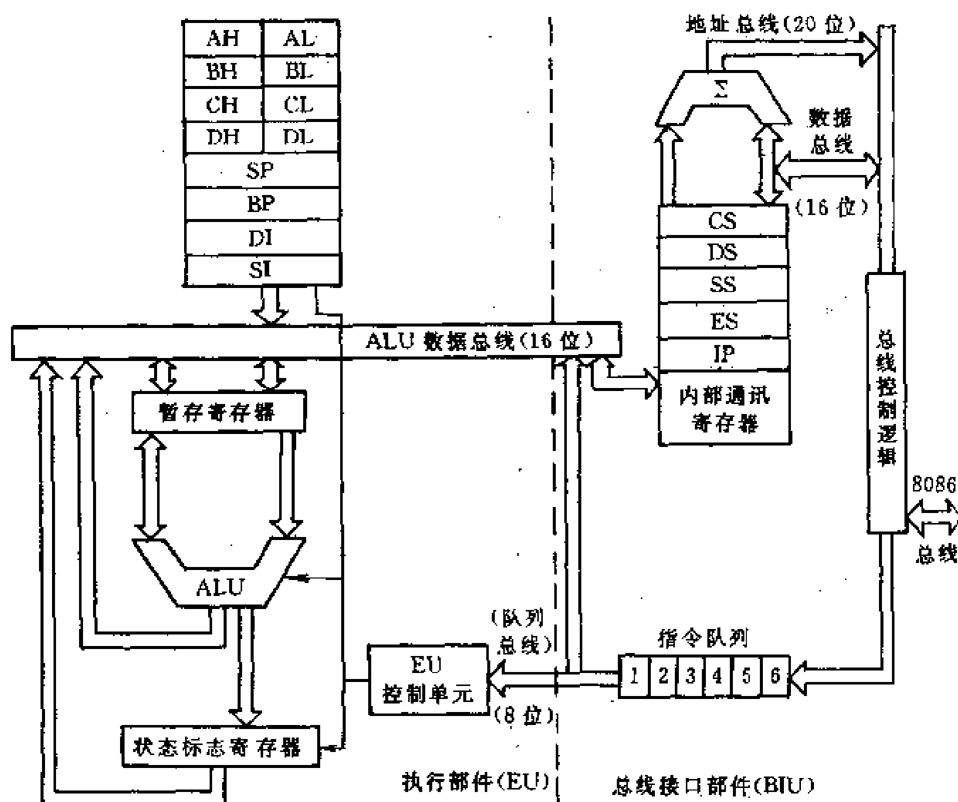


图 3-1　8086CPU 结构基本框图

负责与存储器、I/O 端口传送信息。当 EU 从指令队列中取走指令，指令队列出现空字节时，BIU 即从内存中取出后续的指令代码放入队列中；当 EU 需要数据时，BIU 根据 EU 给出的地址，从指定的内存单元或外设中取出数据供 EU 使用；当运算结束时，BIU 将运算结果送入指定的内存单元或外设。当队列空时，EU 就等待，直到有指令为止。当 8088 队列空出一个字节，8086 空出两个字符时，BIU 就自动执行一次取指令周期，将新指令送入队列。若 BIU 正在取指令，EU 发出访问总线的请求，则必须等 BIU 取指令完毕后，该请求才能得到响应。一般情况下，程序顺序执行，当遇到转移指令时，BIU 就使指令队列复位，从新地址开始取出指令，并立即传给 EU 去执行，其后续指令取来填入指令队列。

指令队列的存在使 8086/8088 的 EU 和 BIU 并行工作，从而减少 CPU 为取指令而等待的时间，提高了 CPU 的利用率，加快了整机的运行速度，另外也降低了对存储器存取速度的要求，这种技术叫并行技术。在整个程序运行期间，BIU 总是忙碌的，充分的利用了总线，效率很高。如图 3-2 所示。

2. 8086/8088 的内部寄存器

如果一个处理器中没有通用寄存器，那么在指令执行的过程中要用到操作数时，必须到存储器中去取，运算的结果也必须立即送到存储器中保留起来。从存储器存取数据要占用总线周期。如果在指令执行的过程中，只要碰到操作数的存取就进行存储器操作，则势必要加长指令的执行时间。如果在处理器中设有一些寄存器，这些寄存器可用来暂时存放参加运算的操作数和运算过程中的中间结果，则在程序执行的过程中就不必每时每刻都要去到存储器中存取数据。在处理器中，用通用寄存器暂时存放操作数可以提高程序执行速度。一般来说，处理器中包含的寄存器越多，处理器使用就越灵活，处理器执行程序的速度也就越快。8086/8088 内部

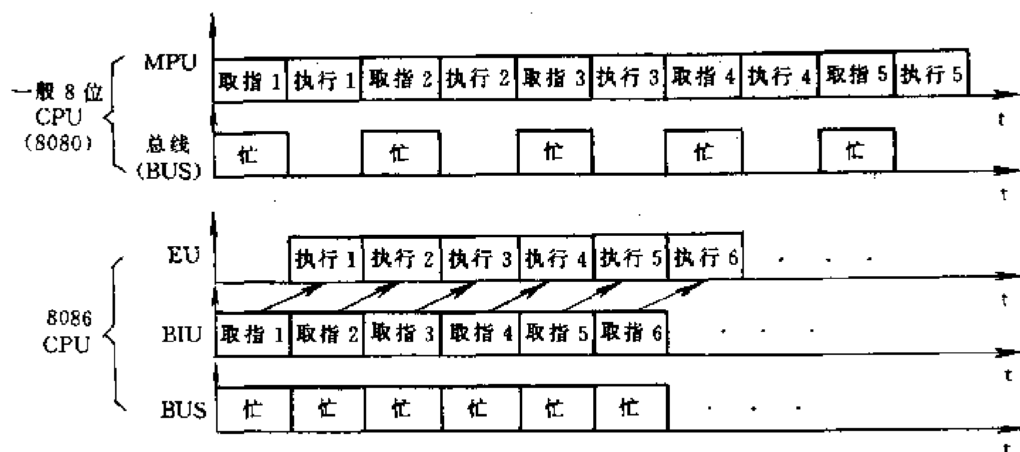


图 3-2 取指令与执行指令时间图

有 14 个 16 位寄存器。按其功能，可分为三大类：第一类是通用寄存器（8 个)。第二类是段寄存器(4 个)，第三类是控制寄存器(2 个)，如图 3-3 所示。

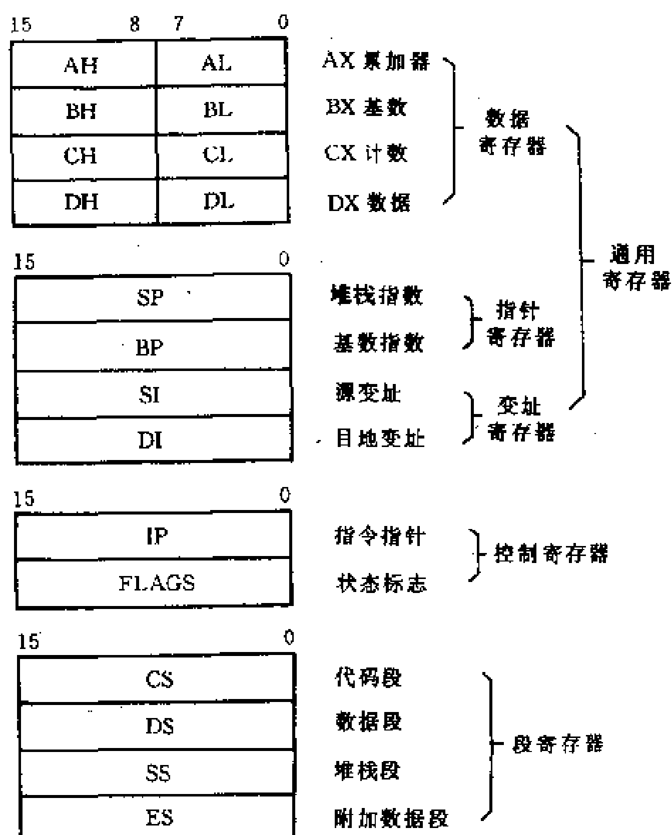


图 3-3 8086/8088 内部寄存器

1）通用寄存器

通用寄存器包括数据寄存器，地址指针寄存器和变址寄存器。

(1) 数据寄存器 AX、BX、CX、DX

数据寄存器一般用于存放参与运算的数据或运算的结果。每一个数据寄存器都是 16 位寄

存器，但又可将高、低 8 位分别作为两个独立的 8 位寄存器使用。它们的高 8 位记作 AH、BH 、CH、DH，低 8 位记作 AL、BL、CL、DL，这给编程带来很大的方便。

上述 4 个寄存器一般作为通用寄存器使用，但它们又有各自的习惯用法。

AX(Accumulator)称为累加器，所有的 I/O 指令都使用该寄存器与外设端口传送信息。BX(Base)称为基址寄存器，在计算内存地址时，常用来存放基址。CX(Count)称为计数寄存器，在循环和串操作指令中用作计数器。DX(Data)称为数据寄存器，在寄存器间接寻址的 I/O 指令中存放 I/O 端口地址，在做双字长乘除法运算时，DX 与 AX 合起来存放一个双字长数据(32 位)，其中 DX 存放高 16 位。

(2) 地址指针寄存器　SP、BP

SP(Stack Pointetr)称为堆栈指针寄存器，BP(Base Pointer)称为基址指针寄存器。作为通用寄存器的一种，它们可以存放数据，但实际上，它们更经常更重要的用途是存放内存单元的偏移地址。

(3) 变址寄存器　SI、DI

SI(Sourse Index)称为源变址寄存器，DI(Destination Index)称为目的变址寄存器，常常用于寻址。

2) 段寄存器　CS、SS、DS、ES

CS(Code Segment)称为代码段寄存器。SS(Stak Ssegment) 称为堆栈段寄存器。DS(Data Segment)称为数据段寄存器，ES(Extra Segment)称为附加数据段寄存器，段寄存器用于存放段基值(16 位的无符号数)。

3) 控制寄存器　IP、FLAGS

IP(Instruction Pointer)称为指令指针寄存器，用于存放预取指令的偏移地址。CPU 从代码段中偏移地址为 IP 的内存单元中取出指令代码的一个字节后，IP 自动加 1，指向指令代码的下一个字节。用户程序不能直接访问 IP。

FLAGS 称为标志寄存器，它是 16 位的寄存器，但只用其中的 9 位。这 9 位包括 6 个状态标志位和 3 个控制标志位，如下所示：

15				11	10	9	8	7	6		4		2		0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

状态标志位记录了算术运算和逻辑运算结果的一些特征，如：结果是否为“0”，是否有进位或借位，结果溢出等。不同的指令对标志位的影响是不同的。

CF　进位标志位。当进行加法或减法运算时，若最高位发生进位或借位，则 CF＝1，否则 CF＝0。

PF　奇偶标志位。当逻辑运算的结果低 8 位中“1”的个数为偶数时 PF＝1，为奇数时 PF＝0。

AF　辅助进位位。在 8(16)位加减法操作中，低 4 位向高 4 位有进位或借位发生时，AF＝1 ，否则 AF＝0。

ZF　零标志位。当运算结果为零时 ZF＝1，否则 ZF＝0。

SF　符号标志位。当运算结果的最高位为 1 时，SF＝1，否则 SF＝0。

OF　溢出标志位。当运算结果超出了带符号数的范围，即溢出时，OF＝1，否则 OF＝0。8

位补码的整数范围是 $-128\sim+127$，16 位补码的整数范围是 $-32768\sim+32767$。

控制标志位　控制标志被设置后，便对其后的操作产生控制作用。

TF　跟踪标志位。TF＝1，使 CPU 处于单步执行指令的工作方式，这种方式便于进行程序的调试，每执行一条指令后，自动产生一个内部中断，从而使用户能逐条指令地检查程序。

IF　中断允许标志位。IF＝1，使 CPU 可以响应可屏蔽中断请求；IF＝0，使 CPU 禁止响应可屏蔽中断请求，IF 的状态对不可屏蔽中断及内部中断没有影响。

DF　方向标志位。DF＝1，使串操作按减地址方式进行，也就是说，从高地址开始，每操作一次地址减少一次；DF＝0，使串操作按增地址方式进行。

二、8086/8088 CPU 的引脚及其功能

8086/8088CPU 均属高性能微处理器，根据它的基本性能至少包含 16 条数据线和 20 条地址线，再加上其它一些必要的控制信号。由于芯片引脚线数量不能太多，因此对部分引脚采用了分时复用方式，构成 40 条引脚的双列直插式封装。8086CPU 封装外形与内部各功能部件之间相互连接如图 3-4 所示。

由于 8086/8088CPU 具有两种工作模式（最小模式和最大模式），8 条引脚（24～31 脚）在两种工作模式中，具有不同的功能，图 3-4(a)括号中所示为最大模式下被重新定义的控制信号。

下面对各引脚功能作简要说明：

1. $AD_{15}\sim AD_0$(Address Data Bus)

分时复用的地址数据线，传送地址时三态输出，传送数据时可双向三态输入/输出。

2. $A_{19}/S_6\sim A_{16}/S_3$(Address/Status)

分时复用的地址状态线，作地址线用时，$A_{19}\sim A_{16}$与 $AD_{15}\sim AD_0$ 一起构成访问存储器的 20 位物理地址。当 CPU 访问 I/O 端口时，$A_{19}\sim A_{16}$保持为“0”。

作状态线用时，$S_6\sim S_3$ 用来输出状态信息，其中：S_3 和 S_4 可用于表示当前使用的段寄存器号，如表 3-1 所示。当 $S_4S_3=10$ 时，表示当前正使用 CS 对存储器寻址，而当前正对 I/O 端口或中断矢量寻址时，不需要使用段寄存器。

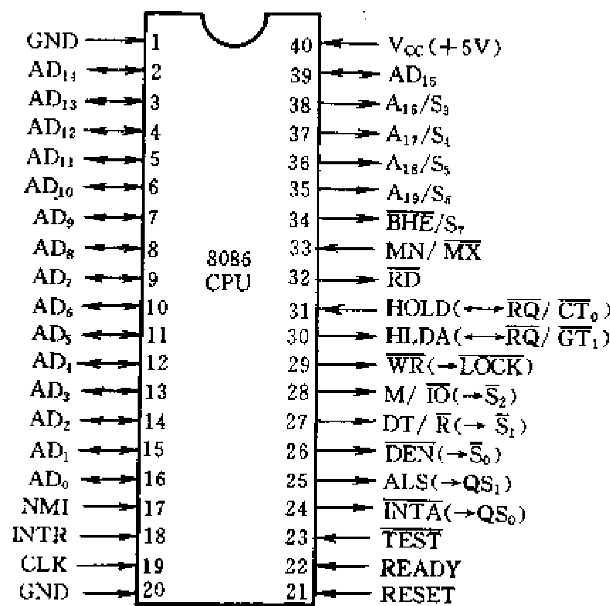
表 3-1　S_4S_3状态编码

S_4	S_3	段寄存器
0	0	ES
0	1	SS
1	0	CS(I/O, INT)
1	1	DS

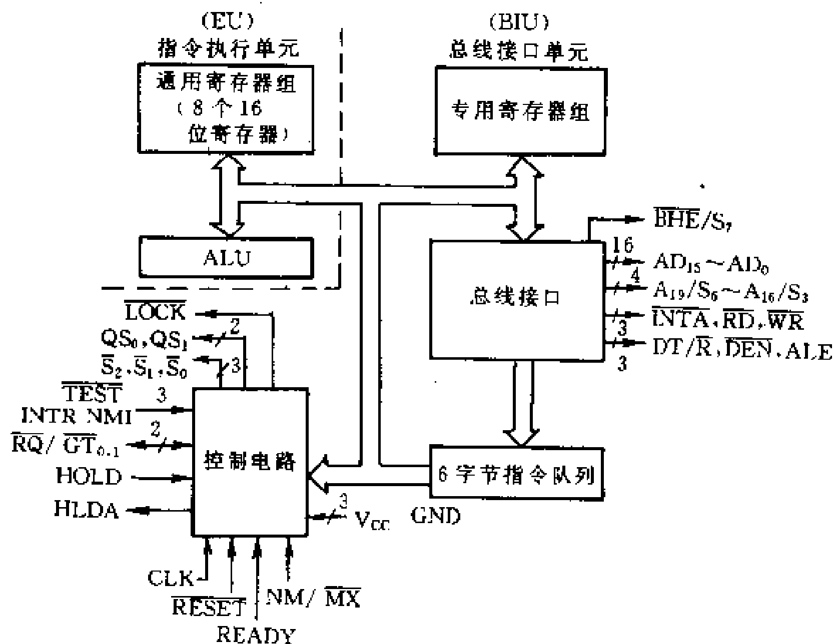
S_5 用来表示中断标志位状态。当 IF＝1 时，S_5 置“1”；S_6 恒保持为“0”。

3. $\overline{BHE}/S_7$(Bus High Enable/Status)

总线高字节有效信号，三态输出，低电平有效。用来表示当前高 8 位数据总线上的数据有效。读/写存储器或 I/O 端口以及中断响应时，$\overline{BHE}$用来作选体信号，与最低位地址码 A_0 配合表示当前总线使用情况，如表 3-2 所示。



(a)封装外型



(b)内部功能部件的相互连接

图 3-4　8086 封装外形与内部功能部件的连接

表 3-2　$\overline{BHE}$和AD_0编码的含义

$\overline{BHE_0}$	AD_0	总线使用情况
0	0	16 位数据总线上进行字传送
0	1	高 8 位数据总线上进行字节传送
1	0	低 8 位数据总线上进行字节传送
1	1	无效

非数据传送期间，输出 S_7 状态信息，在 CPU 处于保持响应期间被设置为高阻状态。

4. $\overline{RD}$(Read)

读信号,三态输出,低电平有效。表示当前 CPU 正在读存储器或 I/O 端口。

5. $\overline{WR}$(Write)

写信号,三态输出,低电平有效。表示当前 CPU 正在写存储器或 I/O 端口。

6. M/$\overline{IO}$(Memory/IO)

存储器或 I/O 端口访问信号,三态输出,M/$\overline{IO}$为高电平时,表示当前 CPU 正在访问存储器;为低电平时,表示 CPU 正在访问 I/O 端口。

7. READY

准备就绪信号,由外部输入,高电平有效,表示 CPU 访问的存储器或 I/O 端口已为传送做好准备。当 READY 无效时,要求 CPU 插入一个或几个等待周期 T_W,直到 READY 信号有效为止。

8. INTR (Interrupt Request)

中断请求信号,由外部输入,电平触发,高电平有效。INTR 有效时,表示外部向 CPU 发出中断请求。CPU 在每条指令的最后一个时钟周期对 INTR 进行测试,一旦测试到有中断请求,并且当前中断允许标志 IF=1 时,则暂停执行下条指令转入中断响应周期。

9. $\overline{INTA}$(Interrupt Acknowledge)

中断响应信号,向外部输出,低电平有效。表示 CPU 响应了外部发来的 INTR 信号,在中断响应周期,可用来作为读选通信号。

10. NMI (Non-Mackable Interrupt Reguest)

不可屏蔽中断请求信号,由外部输入,边沿触发,正跳沿有效。不受中断允许标志的限制,CPU 一旦测试到 NMI 请求有效,待当前指令执行完就自动从中断入口地址表中找到类型 2 中断服务程序的入口地址,并转去执行,显然这是一种比 INTR 高级的请求。

11. $\overline{TEST}$

测试信号,由外部输入。低电平有效。当 CPU 执行 WAIT 指令时,每隔 5 个时钟周期对$\overline{TEST}$进行一次测试。若测试到$\overline{TEST}$无效,则 CPU 处于踏步等待状态,直到$\overline{TEST}$有效,CPU 才继续执行下一条指令。

12. RESET

复位信号,由外部输入,高电平有效。RESET 信号至少保持 4 个时钟周期。CPU 接收到 RESET 信号后,停止进行操作,并将标志寄存器、段寄存器、指令指针 IP 和指令队列等复位到初始状态。

13. ALE (Address Latch Enatle)

地址锁存允许信号,向外部输出,高电平有效。在最小模式系统中用来作地址锁存器 8282/8283 的选通信号。

14. DT/$\overline{R}$(Data Transmit/Receive)

数据发送/接收控制信号,三态输出。在最小模式系统中用来控制数据收发器 8286/8287 的数据传送方向。当 DT/$\overline{R}$ 为高电平时,表示数据从 CPU 向外部输出,即完成写操作,DT/$\overline{R}$为低电平时,表示数据从外部向 CPU 输入,即完成读操作。

15. $\overline{DEN}$(Data Enable)

数据允许信号,三态输出,低电平有效。在最小模式系统中用来作数据收发器 8286/8287 的选通信号。

16. HOLD (Hold Request)

总线请求信号，由外部输入，高电平有效。在最小模式系统中表示有其它共享总线的主控者向 CPU 请求使用总线。

17. HLDA (Hold Acknowlege)

总线响应信号，向外部输出，高电平有效。CPU 一旦测试到有 HOLD 请求时，就在当前总线周期结束时，使 HLDA 有效，表示响应这一总线请求，并立即让出总线使用权。CPU 中指令执行部件(EU)可继续工作到下一次要求使用总线为止，一直到 HOLD 无效，CPU 才将 HLDA 置成无效，并收回对总线的使用权，继续操作。

18. MN/$\overline{MX}$(Minimun/Maximun)

工作模式选择信号，由外部输入。MN/$\overline{MX}$为高电平，表示 CPU 工作在最小模式系统中，MN/$\overline{MX}$为低电平时，表示 CPU 工作在最大模式系统中。

19. CLK (Clock)

主时钟信号，由 8284 时钟发生器输入。8086 CPU 可使用的时钟频率随芯片型号不同而异，8086 为 5MHz，8086－1 为 10MHz，8086－2 为 8MHz。

20. V_{CC}(电源)

8086CPU 只需要单一的＋5 伏电源，由 V_{CC}输入。

下面对 8086CPU 工作在最大模式系统中几个重新定义的引脚作简要的说明。

1. $\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$(Bus Cycle Status)

总线周期状态信号，三态输出。在最大模式系统中由 CPU 传送给总线控制器 8288，8288 对它们译码后代替 CPU 输出相应的控制信号。$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$和具体的操作过程之间的对应关系如表 3-3 所示：

表 3-3　S_2、S_1、S_0的代码组合和对应的操作

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	操　作　过　程
0	0	0	发中断响应信号
0	0	1	读 I/O 端口
0	1	0	写 I/O 端口
0	1	1	暂　停
1	0	0	取指令
1	0	1	读内存
1	1	0	写内存
1	1	1	无源状态

这里，需要对无源状态作一个说明。对$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$来说，在前一个总线周期的 T_4 状态和本总线周期的 T_1T_2 状态中，至少有一个信号为低电平，每种情况下，都对应了某一个总线操作过程，通常称为有源状态。在总线周期的 T_3 和 T_W 状态并且 READY 信号为高电平时，$\overline{S_2}$、$\overline{S_1}$和$\overline{S_0}$都成为高电平。此时，一个总线操作过程就要结束，另一个新的总线周期还未开始，通常称为无源状态。而在总线周期的最后一个状态即 T_4 状态，$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$中任何一个或几个信号改变，都意味着下一个新的总线周期的开始。

2. $\overline{LOCK}$

封锁信号，三态输出，低电平有效。$\overline{LOCK}$有效时表示 CPU 不允许其它总线主控者占用总线。这个信号由软件设置，当在指令前加上$\overline{LOCK}$前缀时，则在执行这条指令期间，$\overline{LOCK}$保持

有效，即在指令执行期间，封锁其它主控者使用总线。

3. $\overline{RQ}/\overline{GT}_0$、$RQ/\overline{GT}_1$(Request/Grant)

请求/同意信号，双向，低电平有效。输入时表示其它主控者请求使用总线；输出时表示CPU对总线请求的响应信号，两条线可同时与两个主控者相连。内部保证$\overline{RQ}/\overline{GT}_0$比$\overline{RQ}/\overline{GT}_1$有较高优先级。

4. QS_1、QS_0(Instruction Queue Status)

指令队列状态，向外部输出。用来表示CPU中指令队列当前的状态，其含义如表3-4所示：

表 3-4 S_1 QS_0编码的含义

QS_1	QS_0	含 义
0	0	无操作
0	1	从队列中取第一个字节
1	0	队列已空
1	1	从队列中取后续字节

8088 CPU 的封装外形如图 3-5 所示。大部分引脚名称及功能与 8086 相同，所不同之处在于：

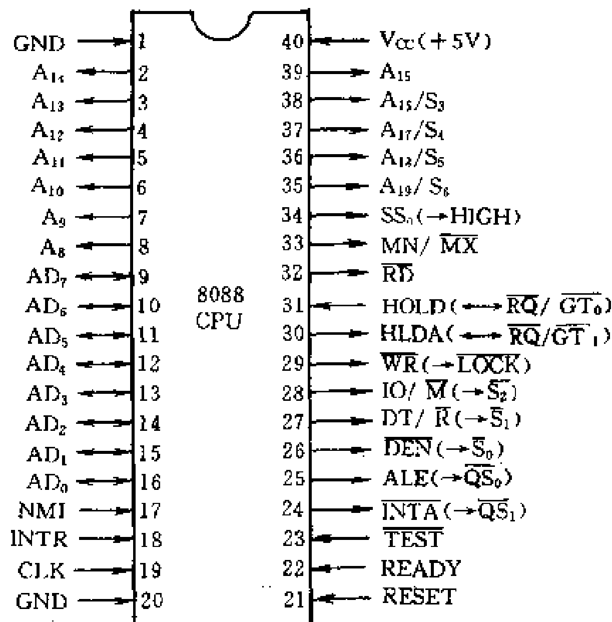


图 3-5 8088CPU 封装外形图

1. 由于 8088 的外形数据总线只需 8 条，因此分时复用地址数据总线只有 $AD_7 \sim AD_0$、$AD_{15} \sim AD_8$ 专门用来传送地址而成为 $A_{15} \sim A_8$。

2. 第 28 号引脚在 8086 中是 $M/\overline{IO}$，在 8088 中改为 $IO/\overline{M}$，定义相反。

3. 第 34 号引脚在 8086 中是$\overline{BHE}$，在 8088 中改为 SS_0，它与 $DT/\overline{R}$、$IO/\overline{M}$ 一起表示 8088 在最小模式中的周期状态。如表 3-5 所示。

表 3-5 SS_0、$IO/\overline{M}$、$DT/\overline{R}$的组合及对应的操作

$IO/\overline{M}$	$DT/\overline{R}$	SS_0	操　　作
1	0	0	发中断响应信号
1	0	1	读 I/O 端口
1	1	0	写 I/O 端口
1	1	1	暂停
0	0	0	取指令
0	0	1	读内存
0	1	0	写内存
0	1	1	无源状态

三、8088 与 8086 的比较

准十六位的 8088CPU 是继 8086 之后推出的，被畅销全球的 IBM－PC 机选作 CPU。它与 8086 CPU 具有类似体系结构，两者的执行部件 EU 完全相同，其指令系统，寻址能力及程序设计方法都相同，所以两种 CPU 完全兼容。这两种 CPU 的区别仅在于 BIU，归纳起来有以下几个方面：

1. 外部数据总线位数的差别：8086 CPU 的外部数据总线有 16 位，在一个总线周期内可以输入/输出一个字(16 位数据)，使系统处理数据和对中断响应的速度加快，而 8088 CPU 的外部数据总线为 8 位，在一个总线周期内只能输入/输出一个字节(8 位数据)。

2. 指令队列的差别：8086 CPU 的指令队列可容纳 6 个字节，且在每一个总线周期中从存储器中取出 2 个字节的指令代码填入指令队列；而 8088 CPU 的指令队列只能容纳 4 个字节，且在每个总线周期内只能取一个字节的指令代码，从而增长了总线取指令的时间，在一定条件下可能影响取指令操作和其它操作的并行性。

3. 引脚的差别：两种 CPU 的引脚功能是相同的，但有以下几点不同：

(1) AD_{15}～AD_0 的定义不同：在 8086 中都定义为地址/数据复用总线；而在 8088 中，由于只需要 8 条数据线。因此，对应于 8086 的 AD_{15}～AD_8 这 8 条引脚定义为 A_{15}～A_8，只作地址线使用。

(2) 34 号引脚的定义不同：在 8086 中定义为$\overline{BHE}$信号；而在 8088 中定义为 SS_0，它与 $DT/\overline{R}$、$IO/\overline{M}$一起用作最小方式下的周期状态信号。

(3) 28 号引脚的相位不同：在 8086 中为 $M/\overline{IO}$；而在 8088 中被倒相为 $IO/\overline{M}$，以便与 8080/8085 系统的总线结构兼容。

第二节　8086/8088 系统总线的构成

一、最小方式下系统总线的构成

在实际使用 8086/8088 微处理器时，还必须配有时钟发生器（8284A），地址锁存器(8282/8283)和总线驱动器(8286/8287)，才能构成系统总线。

1. 时钟发生器 8284A

8284A 除为 CPU 和系统提供时钟信号外，还提供经时钟同步的复位信号 RESET 和就绪信号 READY。

8284A 能为 CPU 提供的最高时钟信号频率为 8MHz。而 8284A－1 可提供 10MHz 时钟信号。为使 8284A 正常工作，只要外接一块晶体即可。

8284A 的引脚图和内部电路框图分别示于图 3-6 和图 3-7 中

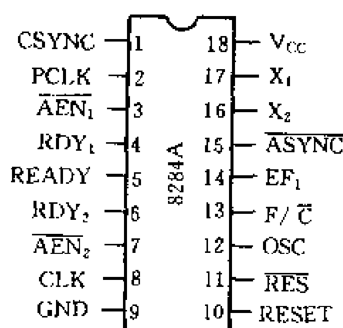


图 3-6 8284A 引脚图

8284A 各引脚的功能如下：

$\overline{AEN_1}$、$\overline{AEN_2}$ 地址允许(输入)

$\overline{AEN}$为低电平有效的信号，用来制约相应的总线就绪信号 RDY。$\overline{AEN_1}$使 RDY 能对 READY 输出信号起作用，而$\overline{AEN_2}$使 RDY_2 起作用。在允许 CPU 访问两个多总线的管理系统总线时，使用这两个$\overline{AEN}$信号，这时$\overline{AEN_1}$(或$\overline{AEN_2}$)同总线裁决器 8289 的相应引脚相连。在非多主控器系统中$\overline{AEN}$应接低电平。

RDY_1、RDY_2 总线就绪(输入)

RDY 是高电平有效信号。该信号来自系统总线上的某一个器件，用来指示该器件已为传送数据准备就绪。

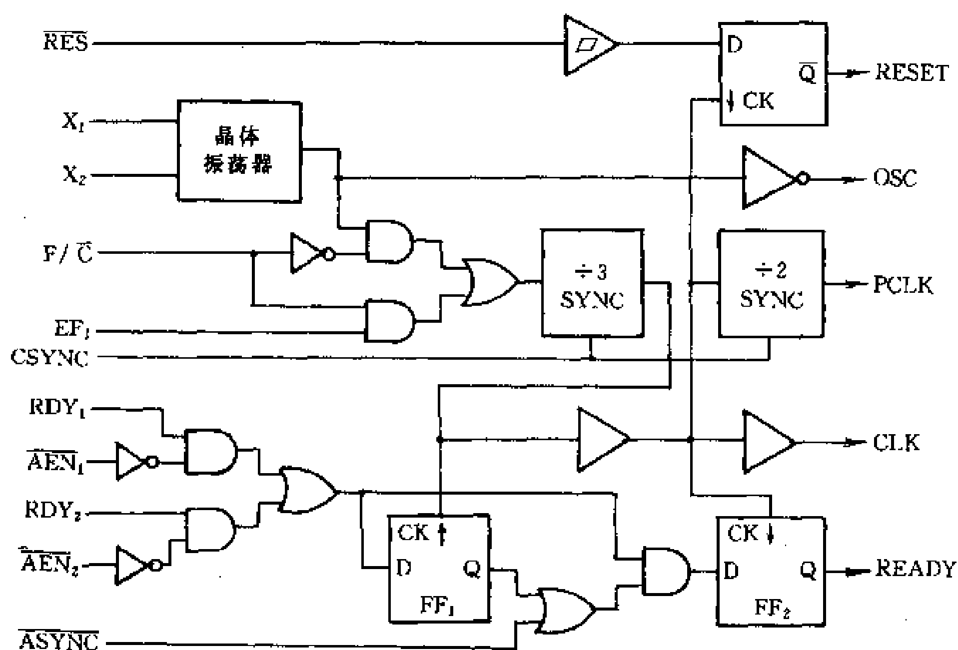


图 3-7 8284A 的内部电路框图

READY 就绪(输出)

就绪信号 READY 为高电平有效。输入信号 RDY_1 或 RDY_2 被时钟同步后形成 READY 信号。

$\overline{ASYNC}$ 就绪同步选择(输入)

$\overline{ASYNC}$输入信号被用来决定 READY 逻辑的同步方式。当$\overline{ASYNC}$为低电平时，8284A 提供 READY 两级同步；而当 ASYNC 置为高电平时，8284A 提供 READY 一级同步。

X1、X2 晶体输入端

X1 和 X2 是连接外接晶体的接线端。晶体频率应是 CPU 时钟频率的 3 倍。

F/$\overline{C}$　频率/晶体选择(输入)

F/$\overline{C}$ 是一个通过接线来选择工作方式的输入引脚信号。当 F/$\overline{C}$ 接低电平时,CPU 的时钟信号由 8284A 内部的晶体振荡器产生;而当 F/$\overline{C}$ 接高电平时,CPU 的时钟信号由 EFI 输入端输入的脉冲信号产生。

EFI　外接频率(输入)

通过此引脚接入外部振荡源信号,该信号应为方波,且其频率应为 CPU 要求的时钟频率的 3 倍。

CLK　处理器时钟(输出)

CLK 时钟信号用作 8086/8088CPU 以及直接与处理器局部总线相连的器件的时钟信号或定时信号。CLK 信号频率是晶体频率或 EFI 输入频率的 1/3,占空比为 33%。其高电平为 4.5V,用于驱动 MOS 器件。

PCLK　外部设备时钟

PCLK 是 TTL 电平信号,其频率是 CLK 时钟频率的 1/2,占空比为 50%。

OSC　振荡器输出

OSC 是内部晶体振荡电路的 TTL 电平输出,其频率等于晶体的频率。

$\overline{RES}$　复位输入

$\overline{RES}$是低电平有效,输入信号,它经内部的施密特触发器和时钟同步后形成 RESET 复位信号。

RESET　复位(输出)

RESET 为高电平有效,用作 8086 系列的处理器的复位信号。

CSYNC　时钟同步(输入)

CSYNC 为高电平有效,用于使系统中的多个 8284A 同步,以提供相位一致的时钟。当 CSYNC 为高电平时,内部计数器被复位,当 CSYNC 变为低电平时,内部计数器重新开始计数。CSYNC 应在外部与 EFI 同步。

2. 8282/8283 地址锁存器

由于 8086/8088CPU 的地址/数据和地址/状态总线是分时复用的,而存储器或 I/O 接口电路通常要求在与 CPU 进行数据传送时,在整个总线周期内须保持稳定的地址信息,因而必须在总线周期的第一个时钟周期内,将地址锁存起来。

8282(不反相)和 8283(反相)是适用于 8086/8088 以及 MCS—85 等系列微型机的 8 位双极型具有三态输出的锁存缓冲器,可用于缓冲或多路传输。图 3-8 和图 3-9 分别是 8282、8283 的逻辑图和引脚图。

8282/8283 具有 8 位数据输入端 $DI_7 \sim DI_0$ 及 8 位数据输出端 $DO_7 \sim DO_0$,当 STB 端的选通脉冲由高电平变为低电平时,$DI_7 \sim DI_0$ 的数据被锁存起来。在选通脉冲的高电平期间,锁存器是"透明的",即锁存器的输出端随出现在输入端的数据而变化。当输出允许信号 $\overline{OE}$为低电平(有效)时,锁存器锁存的数据出现在 8282/8283 的数据输出端 $DO_7 \sim DO_0$ 上,否则,三态缓冲器的输出 $DO_7 \sim DO_0$ 处于高阻状态。在 8086/8088 系列微机中,8282/8283 用来作为地址锁存器,用 ALE 信号作为 8282/8283 的选通脉冲 STB 输入,这样就能在总线周期的第一个时钟期间从地址/数据及地址/状态总线将地址信息锁存于 8282/8283 中,从而保证了在整个总线周期内存储器和 I/O 接口芯片都能获得稳定的地址信息。

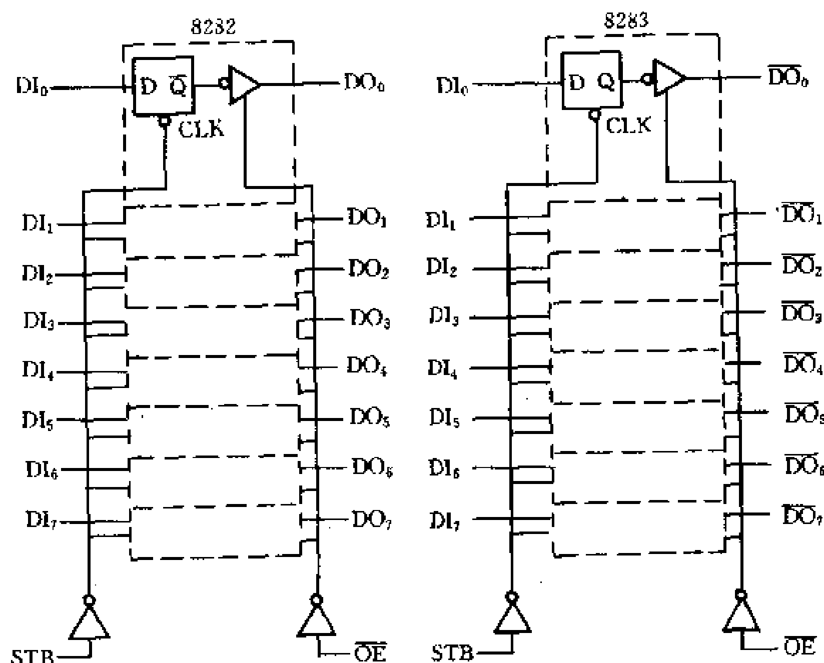


图 3-8　8282/8283 的逻辑图

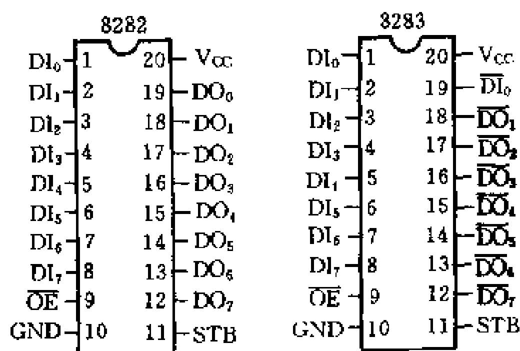


图 3-9　8282/8283 的引脚图

3. 8286/8287 数据收发器

为提高 8086/8088 系统数据总线的驱动能力，并提供一种在多主控器系统应用环境下的控制手段，在 8086/8088CPU 和系统数据总线之间必须接入数据双向缓冲器，INTEL 公司提供的 8286(不反相)和 8287(反相)就是专为这种应用目的而设计的总线数据收发器。

8286/8287 是一种具有三态输出的 8 位双极型数据收发器，具有很强的总线驱动能力。图 3-10 和图 3-11 分别是 8286/8287 的逻辑图和引脚图。

8286/8287 具有 8 路双向缓冲电路，以便实现 8 位数据的双向传送。8286 上的每一路双向缓冲电路都由两个三态缓冲器反向并联组成，8287 的连接方式基本上和 8286 相同，只不过 8287 中的每一个三态缓冲器都有反相作用。

8286/8287 有两个控制输入信号：传送方向控制信号 T 和输出允许信号$\overline{OE}$。它们的控制作用如表 3-6 所示。

在 8086/8088 系列微型计算机系统中，8286/8287 用作数据总线驱动器，其 T 端同 DT/$\overline{R}$ 连接，用于控制数据传送方向，而$\overline{OE}$端同$\overline{DEN}$连接，以保证只在 CPU 需要访问存储器或I/O端口时，才允许数据通过 8286/8287。

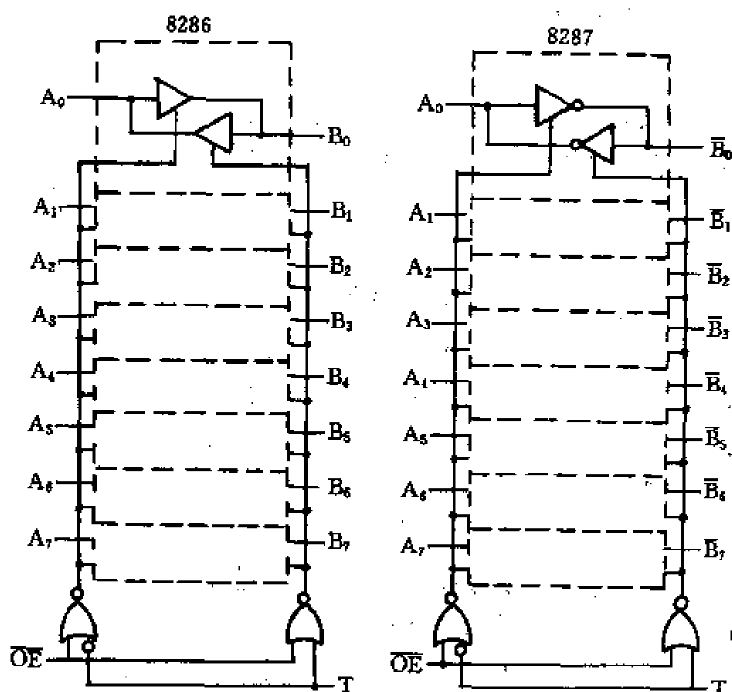


图 3-10 8286/8287 的逻辑图

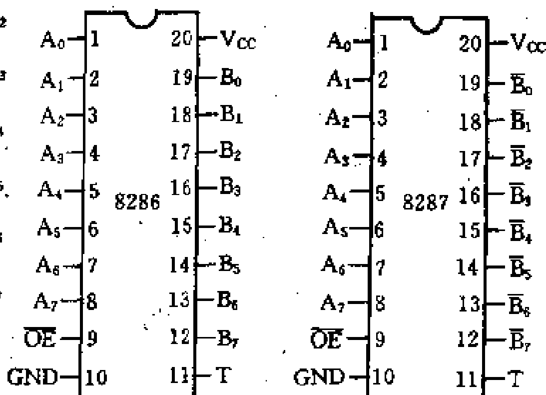


图 3-11 8286/8287 的引脚图

表 3-6 T 和 $\overline{OE}$ 的控制作用

$\overline{OE}$	T	操　　作
0	1	数据从 $A_0 \sim A_7$ 到 $B_0 \sim B_7$
0	0	数据从 $B_0 \sim B_7$ 到 $A_0 \sim A_7$
1	X	$A_0 \sim A_7$，$B_0 \sim B_7$ 均为三态

4. 最小方式系统总线构成

当处理器的 MN/$\overline{MX}$引脚接在＋5V 电源上时，便工作于最小方式。图 3-12 为一种典型的最小方式系统配置。由于只在总线周期的第一部分中提供地址，所以必须把地址锁存起来。通常使用 Intel 8282 作为锁存器。由于 8282 是一种 8 位的锁存器，故锁存 16 位地址线必须用两块，如要锁存所有的 20 位地址线则要用三块 8282。在 8086 系统中，$\overline{BHE}$信号也必须锁存起来。而对较小的系统，由于存储器只有 64K 字节，故只需两块 8282。STB 引脚上的信号用于锁存加在输入数据线 $DI_7 \sim DI_0$ 上的各位数据。因而 STB 与 8086 的 ALE 脚相连，$DI_7 \sim DI_0$ 与 CPU 的 8 位数据线相连。当$\overline{OE}$加上有效的低电平时，将允许锁存器从 $DO_7 \sim DO_0$ 上输出；当$\overline{OE}$上加 1 信号时，将迫使锁存器的输出为高阻状态。在不用 DMA 控制器的 8086 单处理系统中，$\overline{OE}$脚接地。

若一个系统有几个接口，那么这个系统的数据线上就需使用驱动器和收发器，而在小型单板机系统中则不需要驱动器和收发器。图 3-12 中所用的集成电路是 Intel8286 收发器。8286 有 16 个三态元件，8 个驱动器和 8 个接收器。8286 的两组数据引脚是对称的，$A_7 \sim A_0$ 可用于输入，$B_7 \sim B_0$ 用于输出，也可以反过来。输出允许($\overline{OE}$)引脚决定是否允许数据通过 8286，发送引脚(T)控制数据流的方向。当$\overline{OE}$＝1 时，8286 在两个方向上都不能传送数据。$\overline{OE}$＝0 且 T＝1 时，$A_7 \sim A_0$ 为输入。T＝0 时，则 $B_7 \sim B_0$ 为输入。在 8086 系统中，$\overline{OE}$引脚可与处理器的$\overline{DEN}$

引脚相连，每当处理器执行一个 I/O 操作时，$\overline{DEN}$为有效的低电平。$A_7 \sim A_0$ 引脚可与适当的地址/数据线相连接，发送引脚 T 与处理器的 DT/$\overline{R}$ 引脚相连。这样，当处理器输出数据时，数据流从 $A_7 \sim A_0$ 到 $B_7 \sim B_0$，而输入数据则相反。当处理机响应了 HOLD 引脚上的总线请求时，$\overline{DEN}$和 DT/$\overline{R}$ 处于三态。

有时，系统总线设计需将地址和数据信号反相。这样，就又出现了与 8282 和 8286 对应的器件。它们除了输入与输出间倒相外，其余都与 8282 和 8286 一样。与 8282 对应的器件是 8283，而与 8286 对应的器件是 8287。

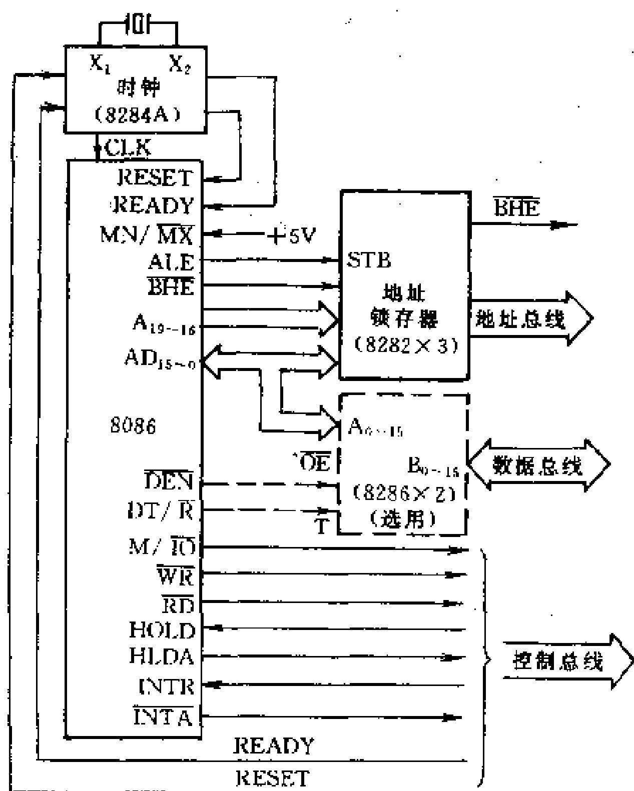


图 3-12 最小方式系统总线

此外，图 3-12 中还出现了 8284A 时钟发生器。这种器件实际上不只是时钟电路，它除了提供频率恒定的脉冲序列外，还用于对准备好(RDY)信号同步，这个信号表示接口已为完成一次传送准备就绪；8284A 还用于同步复位(RES)信号，与时钟信号相结合，对系统进行初始化。尽管系统在任何时刻都可能发 RDY 和 RES 信号，但在时钟后沿到来之前，8284A 是不会让这两个信号到达 READY 和 RESET 引脚上的。

供给 8284 的信号源可来自脉冲发生器(接至 EFI 引脚)，也可来自振荡器(接在 X1 和 X2 间)。如输入 F/$\overline{C}$ 为 1，则由 EFI 输入决定时钟频率；否则，便由振荡器决定时钟频率。不管在哪种情况中，时钟输出 CLK 的频率都是输入频率的三分之一。

上面介绍的三个器件(8282、8286、8284A)都只需要单一+5V 电源，它们的输入和输出都与 TTL 兼容。因而，它们相互间是兼容的且与 8086 也兼容。

在最小系统中，控制线不需要通过收发器，可直接连接。中断响应周期，$\overline{INTA}$信号由两个相邻总线周期内输出的两个负脉冲组成。第一个脉冲通知接口，它的请求已被接受；当接口收到第二个脉冲时，便通过数据总线把中断类型码发给处理器。

二、最大方式下系统总线的构成

1. 8288 总线控制器

前面已指出，当 8086/8088CPU 工作于最大方式时，不直接产生总线控制信号，而是在每个总线周期开始之前输出状态信息$\overline{S_2}$、$\overline{S_1}$和$\overline{S_0}$，用于指示该总线周期的操作类型。8288 总线控制器被用来对$\overline{S_0}$、$\overline{S_1}$和$\overline{S_2}$译码，产生具有适当定时的总线命令和总线控制信号。图 3-13 和 3-14 表示 8288 的内部框图和引脚排列图。

8288 芯片中的状态译码器在时钟发生器的 CLK 信号控制下，对 8086/8088 的状态$\overline{S_2}$、

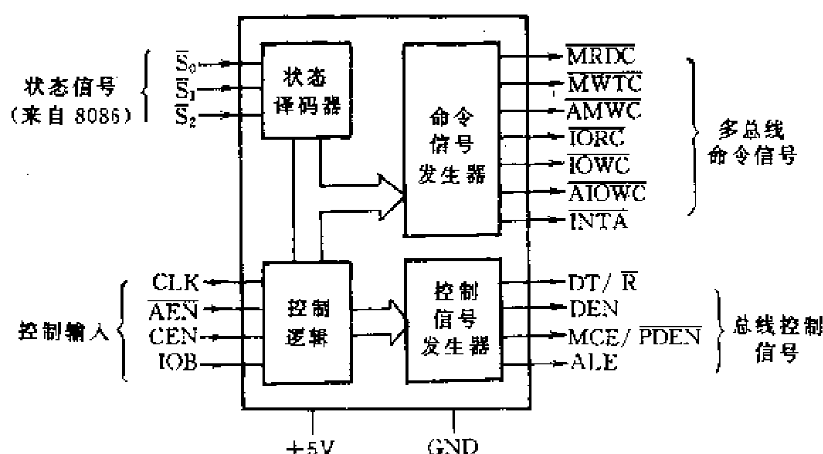


图 3-13　8288 的原理框图

$\overline{S_1}$、$\overline{S_0}$进行译码，产生所需的内部信号。命令信号产生电路和控制信号产生电路再利用上述内部信号产生命令信号和总线控制信号，并且也提高了控制总线的驱动能力。尽管最大方式一般用于多处理器系统，但在一些单处理器系统中，由于此优点，也使用了 8288。

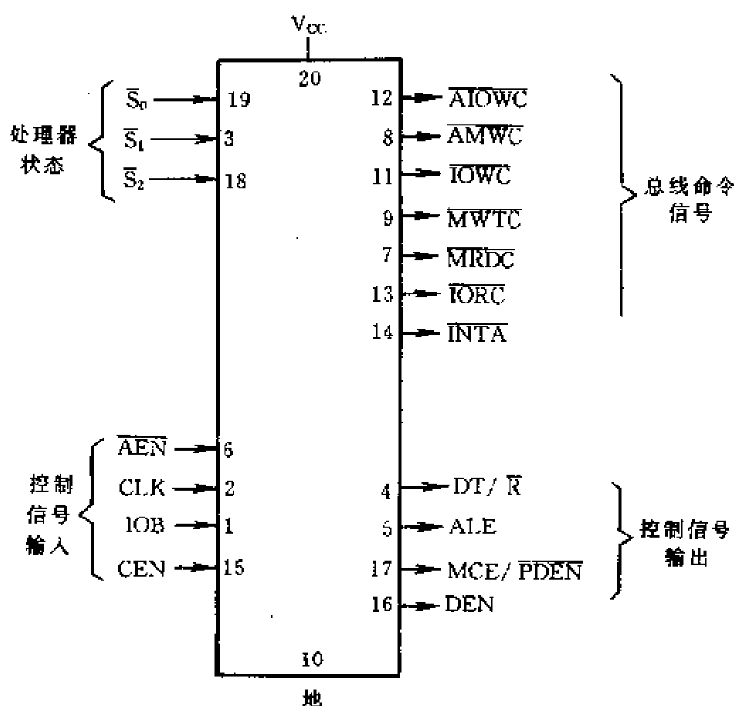


图 3-14　8288 的引脚图

命令信号包括$\overline{MRDC}$(存储器读命令)、$\overline{MWTC}$(存储器写命令)、$\overline{IORC}$(I/O 读命令)、$\overline{IOWC}$(I/O 写命令)、$\overline{INTA}$(中断响应)、$\overline{AMWC}$(提前存储器写命令)以及$\overline{AIOWC}$(提前 I/O 写命令)。在最大方式下，对存储器或 I/O 端口的读、写控制分别以独立的命令信号输出；而在最小方式下，则必须由 M/$\overline{IO}$信号与$\overline{RD}$或$\overline{WR}$信号配合表示上述控制作用。INTA 的功能和最小方式下相同。$\overline{AMWC}$、$\overline{AIOWC}$ 同$\overline{MWTC}$、$\overline{IOWC}$的区别在于前者提前一个时钟周期变为有效(低电平)。这样，对一些较慢的设备或者存储器芯片就得到一个额外的时钟周期去执行写入操作。

除此之外，还有几个信号 ALE、DT/$\overline{R}$ 和$\overline{DEN}$引脚的输出与最小方式时处理机送出的相应信号相同(只是$\overline{DEN}$反相为DEN。CLK 输入，使这个总线控制器的操作与处理器同步。$\overline{AEN}$、IOB 和 CEN 引脚是供多处理机系统使用的。在单处理器系统中，$\overline{AEN}$和 IOB 一般接地，CEN 接高电平。MCE/$\overline{PDEN}$输出是与工作方式有关的，工作方式由加在 IOB 上的信号确定。当 IOB 接地时，其意义为允许设备级联(MCE)，可用于控制级联的 8259A；当 IOB 接 5V 电源

时，其意义为外设数据允许($\overline{PDEN}$)，用于多总线结构中。

CEN (Command ENable) 命令允许信号，由外部输入，高电平有效。

在多片 8288 协同工作时，起控制信号作用。CEN 有效时，允许 8288 输出全部控制信号，CEN 无效，则禁止 8288 发出总线命令信号和总线控制信号中的 DEN 和$\overline{PDEN}$，强制它们呈高阻状态。任何时候只允许一片 8288 的 CEN 信号有效，所以实际上它相当于 8288 芯片的选片信号。单片 8288 使用时，可不提供这个控制信号。

$\overline{AEN}$ (Address Enable) 地址允许信号，由总线仲裁器 8289 输入，低电平有效。

$\overline{AEN}$是支持多总线结构的控制信号，用作多总线之间的同步控制。

MCE/$\overline{PDEN}$(Master Cascade Enable/Peripheral Data Enable) 主控级联允许/外设数据允许信号，向外部输出。

这是一条双功能控制线，当 8288 工作于系统总线方式时，作 MCE 用，在中断响应周期的 T1 状态时 MCE 有效，用来控制主 8259A 向 8259A 输出级联地址，所以被称作主控级联允许信号。当 8288 工作于 I/O 总线方式时，作外设数据允许信号$\overline{PDEN}$用，用来控制外设通过 I/O 总线传送数据。

IOB I/O 总线方式控制信号，输入信号。用来选择 I/O 总线方式或系统总线方式。

如果 IOB 接高电平，则 8288 工作于 I/O 总线方式。这时 CPU 有两条总线：专用的局部 I/O 总线和系统总线，局部 I/O 总线为 8086/8088CPU(以及 8087、8089)所有，系统总线为多个主 CPU 共享。在这种方式下，不管$\overline{AEN}$为何种状态，只要 CPU 有 I/O 访问命令，8288 便输出 I/O 读写命令($\overline{IORC}$、$\overline{AIOWC}$、$\overline{INTA}$)、$\overline{PDEN}$及 DT/$\overline{R}$ 控制信号，后者用于控制局部 I/O 总线的总线收发器 8286/8287，前者用于对接在专用局部 I/O 总线上器件的读/写控制。显然无任何 I/O 读写命令被允许送入系统总线。

当 IOB 接地时，8288 工作于系统总线方式，这时 8288 输出的命令信号用于对系统总线上的存储器和 I/O 接口的读、写控制。在有多个主 CPU 共享系统总线上的存储器和外部设备资源的情况下，8288 的$\overline{AEN}$受总线裁决器 8289 的控制，只有当$\overline{AEN}$为低电平时，才有命令信号产生。

可知 IOB 用来决定 8288 本身的工作方式为：当 IOB 接地时，8288 便工作在适合于单处理器工作的方式，这是 IBM PC/XT 一般情况下设置的状态。此时要求$\overline{AEN}$接地(有效)，CEN 接+5V(有效)。这种方式下，输出端 MCE/$\overline{PDEN}$输出为 MCE 主控级联允许信号。这个信号在含有多个 8259A 的微机系统中，用作系统中主片和从片联络信号 CAS_0、CAS_1、CAS_2 的控制信号。在中断响应的第一个$\overline{INTA}$周期中，MCE 作锁存信号，使作为主片的 8259A 送出的级联地址 CAS_2、CAS_1、CAS_0 进行锁存，以便于在第二个$\overline{INTA}$周期时，用级联地址选中一个从片，并使 CPU 得到中断矢量。

由 8288 总线控制器参与构成的工作于最大方式的单处理器总线如图 3-15 所示。

上述系统中都包含一个主控者——8086CPU 或 8088CPU。平时总线由它们使用，当共享总线的其他主控者，例如 8089IOP 需要使用总线时，8089 应通过$\overline{RQ}/\overline{GT_0}$或$\overline{RQ}/\overline{GT_1}$向 CPU 发出总线请求，待 CPU 接收到这一请求信号，并通过$\overline{RQ}/\overline{GT_0}$或$\overline{RQ}/\overline{GT_1}$回送总线授予信号后，8089 才能使用总线进行数据传送。待数据传送结束，CPU 还要收回总线使用权，而且这种情况下共享总线的处理器数目是有限的。这是因为 CPU 只提供 2 条请求/授予信号线，硬件上保证$\overline{RQ}/\overline{GT_0}$比$\overline{RQ}/\overline{GT_1}$具有较高的优先权。

如果希望在一个系统中能包含多个共享总线的主控者，那么必须在系统中增设总线仲裁

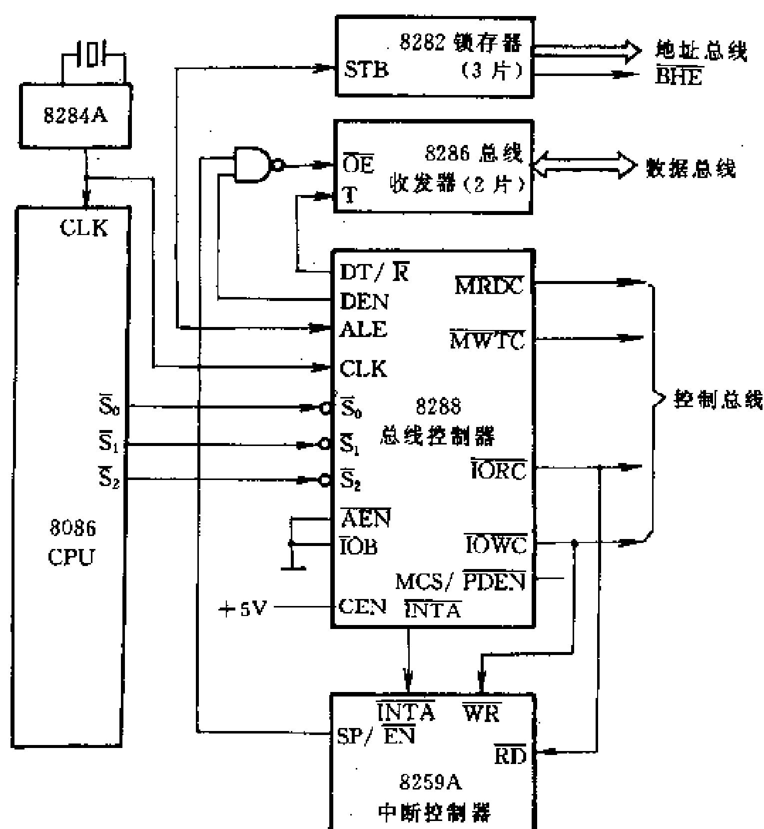


图 3-15 8086CPU 最大模式系统总线

器 8289。

2. 总线仲裁器 8289

总线仲裁器 8289 与总线控制器 8288 配合，将 8086/8088CPU 与 8089IOP、8087NDP 等主控者连接到总线上，构成多主控的微机系统。每一个主控者必须配备一个 8288 总线控制器和一个 8289 总线仲裁器。8289 对主控者是透明的，对于任何一个主控者来说，都好像自己独占总线一样。当有多个主控者同时要求使用总线时，由 8289 总线仲裁器进行仲裁将使用权赋给优先级别高的主控者。

8289 封装外型与内部结构框图如图 3-16 所示。从图中可看出，8289 由状态译码器、控制器、裁决电路和总线接口等几部分构成，为叙述方便，先介绍引脚功能。

(1) 8289 芯片引脚功能

$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$　8086/8088CPU 的总线周期状态信号。8289 将其译码后与其它条件配合完成请求或释放系统总线功能。

$\overline{CBRQ}$　公共总线请求，低电平有效。这是一个双向控制信号，既可输入又可输出，系统中所有 8289 的 CBRQ 端"线或"在一起。对于当前要求占用系统总线的 8289，它是输出信号，$\overline{CBRQ}$有效，表示要求占用总线。对于当前占用系统总线的 8289，它是输入信号，$\overline{CBRQ}$有效，表示有其它处理器要求占用总线。

$\overline{CRQLCK}$　公共总线请求封锁信号，低电平有效，输入信号。$\overline{CRQLCK}$有效时，禁止将总线释放给发出$\overline{CBRQ}$请求信号的主控者。

$\overline{ANYRST}$　任何请求信号，输入信号，高电平有效。$\overline{ANYRQST}$有效时，若$\overline{CBRQB}$也有效，则该 8289 在现行系统总线周期结束时释放系统总线。$\overline{ANYRQST}$无效，则根据给定条件

(a)封装外型

(b)内部结构框图

图 3-16　8289 封装外型与内部结构框图

释放系统总线。

RESB(Resident Bus)　驻留总线信号,输入,高电平有效。RESB 有效时控制该 8289 处于驻留总线方式。它与$\overline{\text{IOB}}$信号配合可选择 8289 的 4 种工作方式之一。

$\overline{\text{IOB}}$　输入输出总线控制信号,输入信号,低电平有效。与 RESB 信号配合来选择 8289 工作方式。

$\overline{\text{AEN}}$　地址允许信号,输出,低电平有效。与总线控制器 8288 的$\overline{\text{AEN}}$直接相连,$\overline{\text{AEN}}$有效时,使 8288 输出总线信号,并立即开启地址锁存器投入工作。释放总线时,8289 使$\overline{\text{AEN}}$无效。

SYSB/$\overline{\text{RESB}}$　系统总线/驻留总线信号,输入。SYSB/$\overline{\text{RESB}}$为高电平,控制主控器处于系统总线方式。SYSB/$\overline{\text{RESB}}$为低电平,则控制主控器处于驻留总线方式。

$\overline{\text{BUSY}}$　总线忙信号,低电平有效,是一个双向信号,即可输入又可输出。系统中所有 8289 的$\overline{\text{BUSY}}$线或起来形成,只有当$\overline{\text{BUSY}}$无效时表示当前总线处于空闲状态,共享总线的主控器才可使用系统总线,并立即将$\overline{\text{BUSY}}$置成有效。这时对正在使用总线的主控者,$\overline{\text{BUSY}}$是输出信号,而对其他主控者 BUSY 是输入信号。

$\overline{\text{BCLK}}$　总线时钟信号。系统中所有的 8289 都由同一个$\overline{\text{BCLK}}$实现同步。

$\overline{\text{BREQ}}$　总线请求信号,输出,低电平有效。当某主控者通过本 8289 请求使用总线时,应使$\overline{\text{BREQ}}$有效,并送往并行优先权裁决电路。

$\overline{\text{BPRN}}$　总线优先权输入信号,低电平有效。$\overline{\text{BPRN}}$=0,表示当前本 8289 具有最高优先级,$\overline{\text{BPRN}}$=1,表示通知本 8289,当前系统总线的使用权已交给其它较高级的 8289。

$\overline{\text{BPRO}}$　总线优先权输出信号,低电平有效。在串行优先权裁决电路中使用,本 8289 输出的$\overline{\text{BPRO}}$接至低一级的 8289 的$\overline{\text{BPRN}}$端。

除此之外还有常规的电源、时钟和地线,8289 的全部引脚名称,符号及传送方向如表3-7所示:

表 3-7　8289 引脚名称

符　　号	名　　称	传送方向
$\overline{S_0}\,\overline{S_1}\,\overline{S_2}$	状态	输入
$\overline{CBRQ}$	公共总线请求	双向
$\overline{CRQLCK}$	公共总线请求封锁	输入
$\overline{ANYRQST}$	任何请求	输入
RESB	驻留总线	输入
$\overline{IOB}$	输入输出总线	输入
$\overline{AEN}$	地址允许	输出
SYSB/$\overline{RESB}$	系统总线/驻留总线	输入
$\overline{BUSY}$	系统忙	双向
$\overline{BCLK}$	总线时钟	输入
$\overline{BREQ}$	总线请求	输出
$\overline{BPRN}$	总线优先权输入	输入
$\overline{BPRO}$	总线优先权输出	输出
CLK	主时钟	输入
V_{CC}	+5V 电源	输入
GND	地	输入

(2)8289 总线工作方式

8289 根据系统的组成可有四种不同的总线工作方式：

①单一总线工作方式

当$\overline{IOB}$和 RESB 均无效时，8289 工作于单一总线方式。这是一种最简单的工作方式，系统中只包含一个主控者，8289 随时监视着主控者的状态线，只要主控者不处于 HALT 状态，并且当前$\overline{BPRN}$有效，$\overline{BUSY}$无效，则主控者就可占用总线，当主控者处于 HALT 状态时，8289 将不请求使用系统总线。

②I/O 总线方式

当$\overline{IOB}$有效，RESB 无效时，8289 处于 I/O 总线方式，这种系统中处理器只通过系统总线访问存储器，一般都以运程方式使用 8289IOP，由 8089 执行的 I/O 程序及其所需要的缓冲区都设在驻留存储器中，在这里驻留存储器当作 I/O 设备处理，I/O 处理器可访问驻留存储器，执行 I/O 程序为外设服务。采用这种方式，允许 IOP 执行驻留存储器中的 I/O 处理程序，主处理器执行系统存储器中的主控程序，两者可并行操作。

③驻留总线方式

当$\overline{IOB}$无效，RESB 有效时，8289 处于驻留总线方式。这种系统中，处理器可访问系统总线也可访问驻留总线。当 SYSB/$\overline{RESB}$端输入高电平时，表示访问系统总线上的存储器或 I/O 设备，这时应请求使用系统总线；当 SYSB/$\overline{RESB}$输入低电平时，应请求使用驻留总线，两条总线上需要设置两个总线控制器 8288。它们分别用来发出各条总线上的访问存储器或 I/O 端口的命令。

④I/O 总线和驻留总线方式

当$\overline{IOB}$有效，RESB 也有效时，8289 上 SYSB/$\overline{RESB}$端输入为高电平，处理器请求使用总

线，只访问系统总线上的存储器，不访问系统总线上的 I/O 端口。若需要访问 I/O 端口时，则可使用 I/O 总线进行。

从而可看出，8289 的四种总线工作方式，取决于$\overline{\text{IOB}}$和$\overline{\text{RESB}}$引线端的状态，如表 3-8 所示：

表 3-8　总线工作方式

$\overline{\text{IOB}}$	RESB	总线工作方式
1	0	单一总线方式
0	0	I/O 总线方式
1	1	驻留总线方式
0	1	I/O 总线和驻留总线方式

(3) 优先权裁决方式

当有多个主控者共享系统总线时，应由 8289 根据条件将系统总线的使用权赋给优先级高的主控者。在不同的系统中可采用三种不同的优先权裁决方式，它们是并行优先权裁决、串行优先权裁决和循环优先权裁决方式。

① 并行优先权裁决方式

采用并行优先权裁决方式的方案之一如图 3-17 所示。

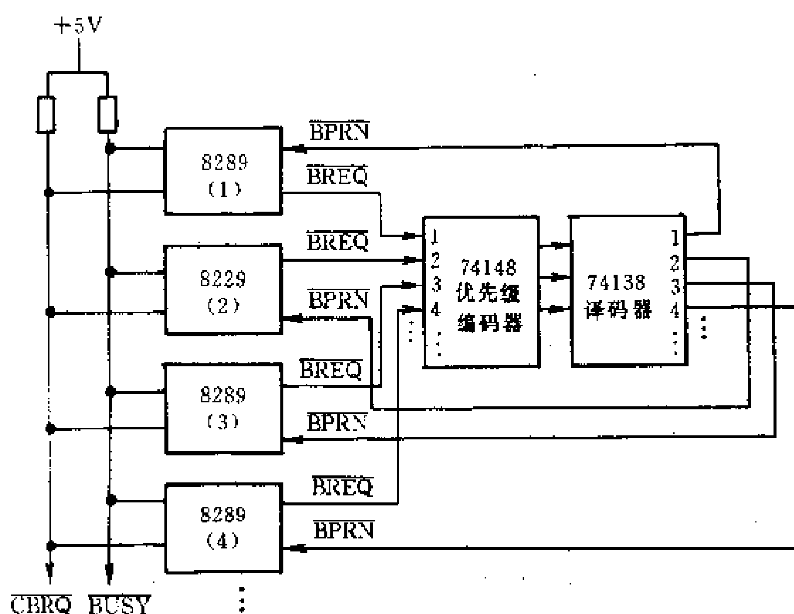


图 3-17　并行优先权裁决电路

采用这种裁决方式需要增设一个优先权编码器和译码器，所有 8289 的总线请求$\overline{\text{BREQ}}$信号并行引入优先权编码器。当 8289 根据来自主控器的状态线和所选用的总线工作方式，判别出主控器需要使用系统总线时，便产生$\overline{\text{BREQ}}$信号有效，经编码器将当前级别最高的$\overline{\text{BREQ}}$的二进制码向译码器输出，再由译码器将该二进制码译为相应的总线优先权输入($\overline{\text{BPRN}}$)送到正在请求使用系统总线的优先级最高的 8289。若当前$\overline{\text{BUSY}}$为高电平，则该 8289 可使用系统总线，并立即将$\overline{\text{BUSY}}$置成有效，以禁止其它 8289 再使用系统总线。

采用这种并行优先权裁决方式的优点是速度快，但需要增加设备。

②串行优先权裁决方式

采用串行裁决方式的方案之一如图 3-18 所示。

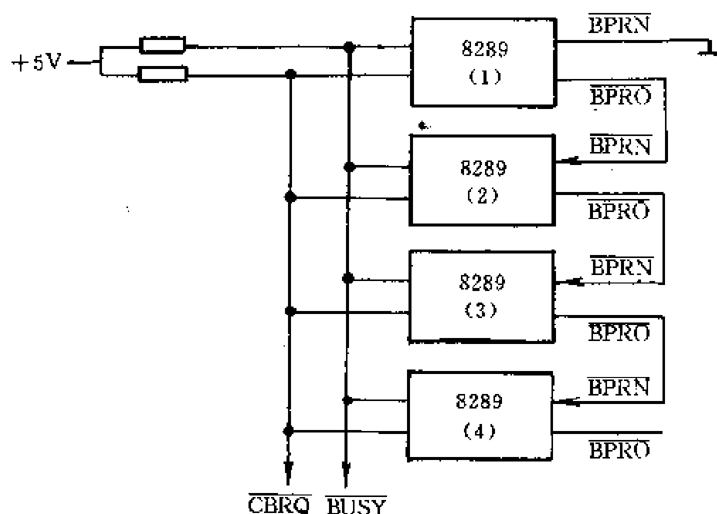


图 3-18　串行优先权裁决电路

采用这种优先权裁决电路不需要增加任何设备，只需将各个 8289 通过总线优先权输入$\overline{BPRN}$和总线优先权输出$\overline{BPRO}$信号串行链接起来。将优先级最高的 8289 的$\overline{BPRN}$端接地，在总线空闲时，所有 8289 的$\overline{BPRN}$和$\overline{BPRO}$均为低电平。表示比它优先级高的主控者没有使用总线。当任何一个 8289 要求使用总线时，只要它的$\overline{BPRN}=0$，且当前$\overline{BUSY}=1$，则它可获得总线使用权，并立即输出$\overline{BPRO}$为 1，然后顺序将优先级比它低的所有 8289 的$\overline{BPRN}$置成高电平，而且使$\overline{BUSY}$置成有效，表示总线忙，禁止其它主控者使用总线。这种裁决方式虽然省设备，但响应速度受到限制。从最高优先级的 8289 输出$\overline{BPRO}$到最低优先级的$\overline{BPRN}$输入的延迟时间不能超过总线时钟周期，在$\overline{BCLK}$最高频率为 10MHz 情况下，最多允许链接 3 片 8289。

③循环优先权裁决方式

采用这种方式的电路结构与并行裁决方式相似，只是优先权编码器电路要复杂一点。每响应一次请求，就将赋给这一级最低优先级，最高优先级赋给原来比它低一级的 8289。其它级的优先权按循环方式改变，这种裁决方式能使得各个 8289 具有平等的使用总线的权利，而不像前面两种方式，优先级高的 8289 总是具有优先使用总线的权利。

(4) 8289 的应用

8289 采用驻留总线方式的系统如图 3-19 所示

该系统中，8086CPU 可通过系统总线访问存储器或 I/O 端口，也可通过 I/O 局部总线访问存储器或 I/O 端口，因此需要设置两个总线控制器 8288，分别发出两条总线上的命令，允许多个主控者共享系统总线，由总线仲裁 8289 完成总线仲裁功能。

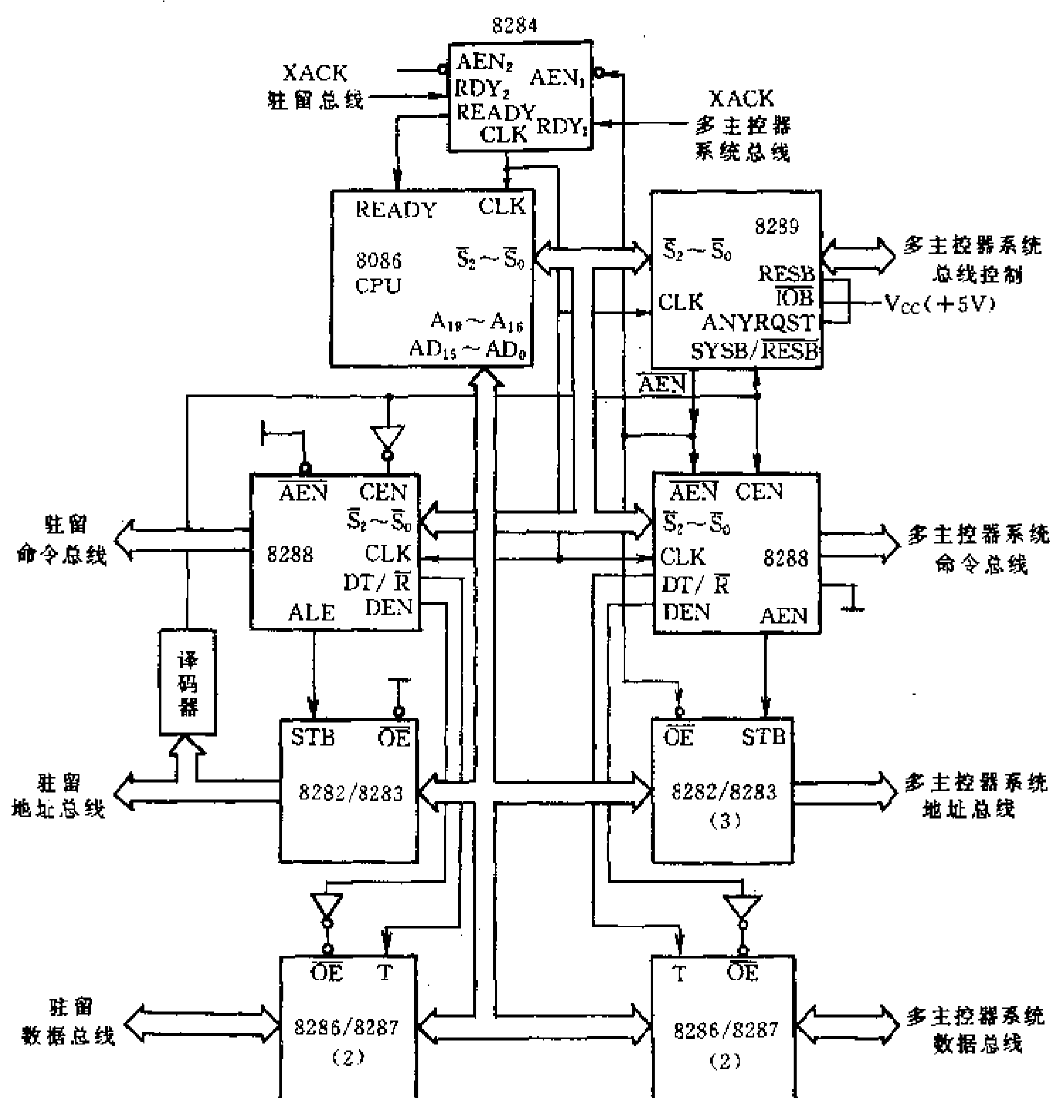


图 3-19　8289 的应用举例

第三节　存储器和 I/O 的组织

80X86 系统为了向上兼容，必须能按字节进行操作，因之系统中存储器和 I/O 端口是按字节编址的。

一、存储器的组织

1. 存储空间与存储器结构

8086/8088 有 20 根地址线，因此，具有 $2^{20}=1M$ 字节的存储器地址空间。这 1M 字节的内存单元按照 00000～FFFFFH 来编址。

(1) 8086 系统中存储器的结构

8086 系统中，将 1M 字节的存储空间分成两个 512K 字节的存储体，一个存储体中包含偶数地址，另一个存储体中包含奇数地址，两个存储体之间采用字节交叉编址方式，如图 3-20 所

示。

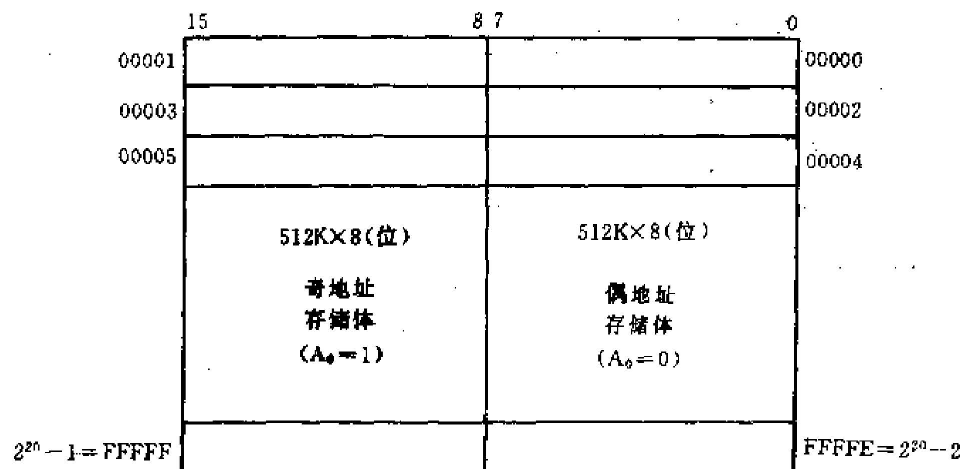


图 3-20　字节交叉编址方式

对于任何一个存储体(偶地址存储体或奇地址存储体),只需要 19 位地址码(A_{19}~A_1) 就够了,另一位地址码(最低位 A_0)可用来区分当前是访问哪一个存储体,也就是说 $A_0=0$,表示访问偶地址存储体;$A_0=1$,表示访问奇地址存储体。但由于 8086 系统中,不仅允许访问存储器读/写一个字节信息,而且也允许访问存储器读/写一个字信息(相邻两个字节),这时要求同时访问两个存储体,各取出一个字节信息。这种情况下若只用 A_0 的取值来控制读/ 写操作就不够了。为此在该系统中,另增设一个总线高位有效控制信号$\overline{BHE}$。当$\overline{BHE}$有效时,选定奇地址存储体,体内地址由 A_{19}~A_1 确定。当 $A_0=0$ 时,选定偶地址存储体,体内地址同样由 A_{19}~A_1 确定,而且偶地址存储体固定与低 8 位数据总线(D_7~D_0)相连, 因此又可称它为低字节存储体。奇地址存储体固定与高 8 位数据总线(D_{15}~D_8)相连,因此,又可称它为高字节存储体。允许 CPU 访问任何一个存储体读/写一个字节信息或同时访问两个存储体读/写一个字的信息。$\overline{BHE}$和 A_0 的控制作用如表 3-9 所示。

表 3-9　$\overline{BHE}$和 A_0 的控制作用

$\overline{BHE}$	A_0	操　　作
0	0	同时访问两个存储体,读/写一个字的信息
0	1	只访问奇地址存储体,读/写高字节的信息
1	0	只访问偶地址存储体,读/写低字节的信息
1	1	无操作

两个存储体与总线之间的连线,如图 3-21 所示。

在 8086 系统中,存储器的这种分体结构对用户来说是透明的。当用户需要访问存储器中的某个字节时,指令中的地址码经变换后应得到 20 位的物理地址,这物理地址当然可以是偶地址,也可以是奇地址。如果是偶地址($A_0=0$),$\overline{BHE}=1$,这时可由 A_0 选定偶地址存储体,由 A_{19}~A_1,从偶地址存储体中选定某个字节地址,并启动该存储体,读/写该地址中一个字节的信息,通过数据总线的低 8 位传送数据;如果是奇地址($A_0=1$), 则偶地址存储体不会被选,也就不会启动它。为了启动奇地址存储体,系统将自动产生$\overline{BHE}=0$, 作为奇地址存储体的选体信号,与 A_{19}~A_1 一起选定奇地址存储体中的某个字节地址,并读/写该地址中一个字节的信

息，通过数据总线的高 8 位传送数据。

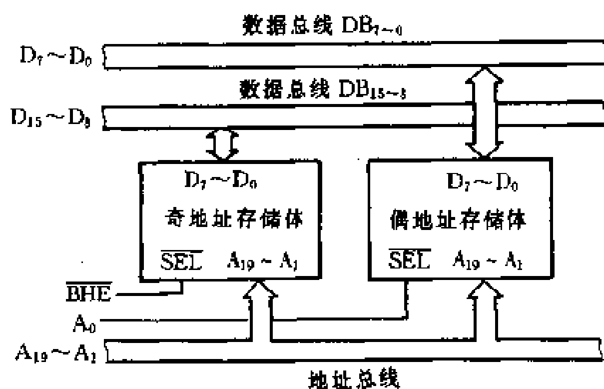


图 3-21 存储体与总线的连接

如果用户需要访问存储体中的某个字(16 位信息)，那么要分两种情况来讨论。一种情况是若用户需要访问的是从偶地址开始的两个字节(即高字节在奇地址中，低字节在偶地址中)，则可一次访问存储器读/写一个字的信息，这时应使 $A_0=0$，$\overline{BHE}=0$。另一种情况是若用户需要访问的是从奇地址开始的两个字节(即高字节在偶地址中，低字节在奇地址中)，这时需要两次访问存储器才能读/写这个字的信息。第一次访问存储器读/写奇地址中的字节，第二次访问存储器读/写偶地址中的字节。显然，为了加快程序的运行速度，希望访问存储器的字地址为偶地址，通常将这种从偶地址开始的字称作"对准字"，而从奇地址开始的字称为"非对准字"。

(2) 8088 系统中存储器的结构

8088 系统中，可直接寻址的存储器空间同样为 1M 字节，但是整个 1M 字节的存储空间同属一个单一的存储体，它与总线之间的连接方式如图 3-22 所示。

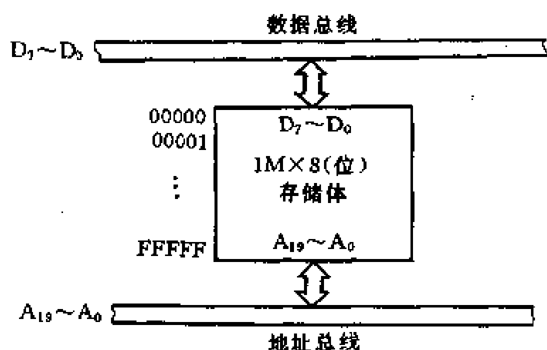


图 3-22 8088 系统中存储器与总线的连接

8088CPU 每访问一次存储器只能读/写一个字节信息，因此 8088 系统的存储器中不存在对准存放的概念，任何数据字都需要两次访问存储器才能完成读/写操作。因此，在 8088 系统中，程序运行速度会比在 8086 系统中慢些。

2. 存储器的分段

8086/8088 系统中，可寻址的存储器空间达 1M 字节，要对整个存储器空间寻址，需要 20 位长的地址码，而机内所有的寄存器都只有 16 位，只能寻址 64K 字节。因此在 8086/8088 系统中，把整个存储空间分成许多逻辑段，这些逻辑段容量最多可为 64K 字节。

8086/8088 系统中，对 1M 字节的存储器采用了一种巧妙的分段办法。将整个存储器分成许多逻辑段，每个逻辑段的容量≤64K 字节，允许它们在整个存储空间中浮动，各个逻辑段之间可以紧密相连，也可以相互重迭(完全重迭或部分重迭)，那么在整个存储空间中可设置若干个逻辑段，如图 3-23 所示。

对于任何一个物理地址来说，可以唯一地被包含在一个逻辑段中，也可包含在多个相互重迭的逻辑段中，只要能得到它所在段的首地址和段内的相对地址，就可对它进行访问。在 8086/8088 存储空间中，把 16 字节的存储空间称作一节(Paragraph)，为了简化操作，要求各个逻辑段从节的整数边界开始，也就是说保证段首地址的低 4 位地址码总是为"0"。于是将段首地址的高 16 位地址码称作"段基值"，把它存放在相应的段寄存器中；而段内的相对地址，可以用系统中的 16 位通用寄存器来存放，被称作"偏移量"。

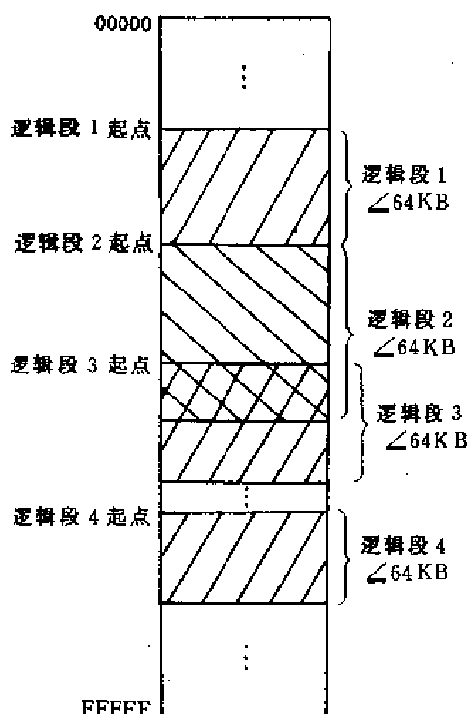


图 3-23　存储器分段示意图

使用段寄存器的优点是：

(1) 虽然各条指令使用的地址只有 16 位，但整个 CPU 的存储器寻址范围可达 1M 字节。

(2) 如果使用多个代码、数据或堆栈段，可使一个程序的指令、数据或堆栈部分的长度超过 64K 字节。

(3) 为一个程序及其数据和堆栈使用独立的存储区提供了方便。

(4) 能够将某个程序及其数据在每次执行时放入不同的存储区域中。

3. 存储器中的逻辑地址和物理地址

采用分段结构的存储器中，任何一个逻辑地址由段基值和偏移地址两个部分构成，它们都是无符号的 16 位二进制数。

任何一个存储单元对应一个 20 位的物理地址，也可称为绝对地址，它是由逻辑地址变换得来的。当 CPU 需要访问存储器时，必须完成如下的地址运算：

$$物理地址 = 段基值 \times 16 + 偏移地址$$

这是在 CPU 的总线接口部件 BIU 的地址加法器中完成的，其操作如图 3-24 所示。

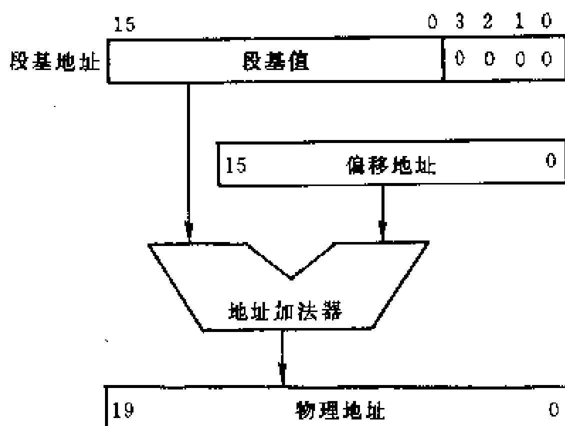


图 3-24　物理地址的形成过程

如果访问存储器要求取指令，则段基值来源于代码段寄存器 CS，偏移量来源于指令指针 IP。

如果访问存储器要求读/写操作数，则通常由数据段寄存器 DS 给出段基值(必要时可指定为 CS、ES 和 SS)，而其偏移地址要由 CPU 的指令执行部件 EU，根据指令中所给出的寻址方式来进行计算，通常将这样计算得到的偏移地址称作"有效地址"(EA)。但如果所采用的寻址方式是通过基址指针寄存器 BP 寻址，则段基值要由堆栈段寄存器 SS 提供(必要时可以指定为 CS、DS 或 ES)。

如果访问存储器对堆栈进行操作，则段基址来源于堆栈段寄存器 SS，偏移地址来源于堆栈指针 SP。

如果执行的是串运算指令，当取源串时，段基值由数据段寄存器 DS 提供（必要时可指定为 CS、ES 和 SS），偏移地址由源变址寄存器 SI 提供，不允许修改；当取目标串时，段基值必须由附加段寄存器 ES 提供，不允许修改，偏移地址由目标变址寄存器 DI 提供，不允许修改。以上这些是系统内部的约定，程序设计过程中必须遵守这些约定，如表 3-10 所示。

表 3-10　逻辑地址来源

操作类型	段基址		偏移地址
	正常来源	其它来源	
取指令	CS	无	IP
堆栈操作	SS	无	SP
存/取变量	DS	CS、ES、SS	有效地址 EA
取源串	DS	CS、ES、SS	SI
存/取目标串	ES	无	DI
通过 BP 间接寻址	SS	CS、ES、SS	有效地址 EA

例如，若某内存单元处于数据段中，DS 的值为 8915H，偏移地址为 0100H。那么这个单元的物理地址为：89150H＋0100H＝89250H。

内存单元逻辑地址一般表示为：段基值：偏移地址。上例中，89250H 单元的地址可写为：8915H：0100H。

存储器分段管理的方法虽给编程带来一些麻烦，但给模块化程序，多道程序及多用户程序的设计创造了条件。

4. 堆栈段的使用

8086/8088 系统中的堆栈是用段定义语句在存储器中定义的一个堆栈段，和其它逻辑段一样，它可在 1M 字节的存储空间中浮动，其容量可达到 64KB，这是一个向下生长的堆栈，由堆栈段寄存器 SS 给定堆栈段的段基值，由堆栈指针 SP 给定当前栈顶，当堆栈置空时，SP 指向堆栈栈底。

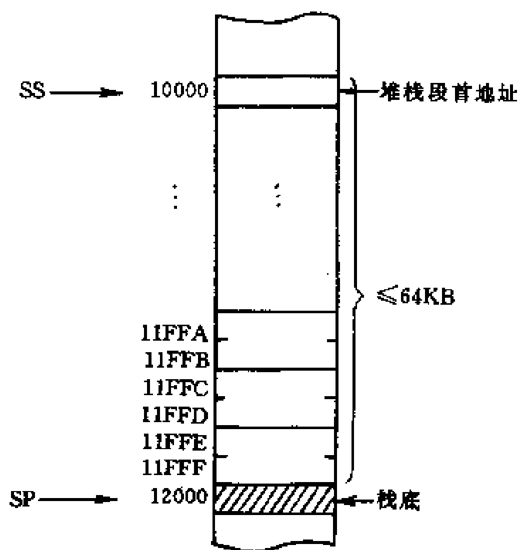


图 3-25　堆栈在存储器中的分布情况

若已知当前(SS)＝1000H，(SP)＝2000H，那么堆栈在存储器中的分布情况如图 3-25 所示。

为了加快堆栈操作的速度，堆栈以字为单位进行操作，并且堆栈中的数据项必须对准存储（低字节在偶地址、高字节在奇地址），以保证每访问一次堆栈，能压入/弹出一个字信息。对堆栈的操作主要是入栈操作(PUSH)和弹出操作(POP)。

进行入栈操作时，总是先修改指针(SP-2→SP)，后将信息入栈。进行弹出操作时，总是先将信息弹出而后修改指针(SP＋2→SP)。

5. 段寄存器初始化

如果运行一个与操作系统相联系的程序，则该操作系统一般先进行段寄存器的初始化，继而当需要时修改它们的内容。如果进行一个没有利用操作系统的程序，那么段寄存器必须初始化。方法一是通

过立即数对段寄存器进行初始化。程序如下：

```
MOV AX,IMM$DATA$FOR$DS ;立即数装入 AX
MOV DS,AX
MOV AX,IMM$DATA$FOR$ES ;立即数装入 AX
MOV ES,AX
MOV AX,IMM$DATA$FOR$SS ;立即数装入 AX
MOV SS,AX
```

初始化段寄存器的另一方法是直接从存储器传送数据到段寄存储器中，程序如下：

```
MOV DS,CS:DATA$FOR$DS
MOV ES,CS:DATA$FOR$ES
MOV SS,CS:DATA$FOR$SS
```

如果段寄存器 ES 和 SS 初始化的数据是包含在由 DS 寻址的段中，则上面这段程序中的第二条及第三条指令中段的前缀可以省去。

对于特殊的初始化程序，8086 提供了专门的保护措施。当 SS 和 SP 寄存器由连续的 MOV 指令初始化时，在两条 MOV 指令之间 8086 不允许中断发生。因此

```
MOV SS,CS:DATA$FOR$SS
MOV SP,DATA$FOR$SP
```

是一个不可中断的指令序列。

二、8086/8088 的 I/O 组织

8086/8088 系统和外部设备之间都是由 I/O 接口电路来联系的。每个 I/O 接口都有一个端口或几个端口。在微机系统中每个端口分配一个地址号，称为端口地址。一个端口通常为 I/O 接口电路内部的一个寄存器或一组寄存器。

8086/8088CPU 用地址总线的低 16 位作为对 8 位 I/O 端口的寻址线，所以 8086/8088 系统可访问的 8 位 I/O 端口有 65536(64K)个。两个编号相邻的 8 位端口可以组成一个 16 位的端口。一个 8 位的 I/O 设备既可以连接在数据总线的高 8 位上，也可以连在数据线的低 8 位上。一般为了使数据地址总线的负载相平衡，希望接在高 8 位和低 8 位的设备数目最好相等。当一个 I/O 设备接在数据地址总线低 8 位(AD_7～AD_0)上的时候。这个 I/O 设备所包括的所有端口地址都将是偶数地址(即 $A_0=0$)；若一个 I/O 设备是接在数据总线的高 8 位(AD_{15}～AD_8)时，那么此设备包含的所有端口地址都是奇数地址(即 $A_0=1$)。如果某种特殊 I/O 设备既可使用偶地址又可使用奇地址时，那么 A_0 就不能作为这个 I/O 设备内部端口的地址选择线使用，此时 A_0 和$\overline{BHE}$这两个信号必须结合起来作为 I/O 设备选择线，用以防止对 I/O 设备的错误操作。

8086CPU 对 I/O 设备的读/写操作与对存贮器的读/写操作相似。当 CPU 与偶地址的I/O 设备实现 16 位的存取操作时，可以在一个总线周期内完成存取操作。当 CPU 与奇地址的I/O 设备实现 16 位的存取操作时，要占用二个总线周期才能完成。

当 8086/8088CPU 接成最小方式工作时，CPU 的读/写命令($\overline{RD}$、$\overline{WR}$)对存储器和 I/O 设备是公用的。如果存储器和 I/O 设备所占用的地址空间是相互重叠的，那么可以通过 M/$\overline{IO}$信号来区分是对存储器读/写操作还是对 I/O 读/写操作。如果系统中存储器和 I/O 设备所占用的地址空间没有重叠，或者当 I/O 设备与存储器在一起统一编址时，可以不受 M/$\overline{IO}$信号的控

制。这样就可以用对存储器的访问指令来实现对 I/O 端口的读/写。由于存储器的读/写指令的寻址方式很多，功能很强，故使用起来较灵活。

在最大模式工作时，若将存储器读/写信号接到 I/O 设备的控制输入端，则此时 I/O 设备地址和存储器地址统一编址；若将 I/O 读/写信号接到 I/O 设备的控制输入端，则此时仅指明是 I/O 设备地址，应由 I/O 指令来完成 I/O 操作。

还应指出，IBM PC 系统只使用了 $A_0 \sim A_9$ 十条地址线作为 I/O 端口的寻址信号，故最多可有 1024 个端口地址。

三、80386/80486 系统的存储器结构

80386、80486 中的基本存储单元是一个字节。一个字(16 位)被存储在存储器的两个连续单元中，低字节在低地址，高字节在高地址。一个双字(32 位) 被存储在存储器的连续四个单元中，最低字节存放在最低地址，最高字节存放在最高地址。一个字或双字的地址，是由它们的最低字节的地址指定的。

除了这些基本的数据类型，80386、80486 还支持两种更大的存储单位：段和页。

存储器分段是 8086、80286 中所采取的数据结构。80386、80486 的存储器可以分为一个或多个可变长度的段，它们可以与磁盘相交换，或由若干程序所共享。这对于按逻辑模块组织存储器系统分段是很有用的，因此成为应用程序员的工具。

80386、80486 还支持分页，存储器还可以组织成一个或多个 4K 字节的页。分页对系统程序员管理一个系统的物理存储器来说是很有用的，特别是在多用户，多任务操作系统中。80386 中分段和分页可以结合起来使用，从而可以兼收并蓄两种系统的优点，为系统程序设计员提供最大的灵活性和方式。

1. 地址空间

80386、80486 有 3 种不同的地址空间：逻辑地址、线性地址和物理地址。逻辑地址(也称为虚拟地址)由一个选择器(段选择器)和一个偏移量组成。选择器是某一个段寄存器的内容，偏移量即是由寻址方式中所有寻址元素(基址、变址、位移量)相加而成的。由于 80386 、80486 的每个任务都有一个最大为 16K($2^{14}-1$)字节的选择器，而偏移量是 32 位的，故可以为 4G(2^{32} 即 4 千兆)字节，从而每个任务可以有总共为 2^{46}即 64T(64 兆兆)字节的逻辑地址空间。

分段部件将逻辑地址空间转换为 32 位线性地址空间。如果不允许分页部件操作，则这个 32 位的线性地址就与物理地址相对应。分页部件将线性地址空间转换为物理地址空间。物理地址就是在芯片地址引出脚上所出现的地址。

实地址方式和保护方式的主要区别就在于分段部件如何完成逻辑地址到线性地址的转换。在实地址方式下，段寄存器中的内容就是段基值(与 8086 中相同)，分段部分将它左移 4 位形成 20 位的段基地址，然后加上 16 位的偏移量从而形成线性地址。而在保护方式下，每个选择器都有一个 32 位的线性基地址与之相联系，段寄存器的内容中有一个索引，由它可以从表中(全局描述表或局部描述表)读出相应的线性地址，与偏移量(32 位)相加从而形成最后的线性地址。

各种地址空间之间的关系如图 3-26 所示。

2. 段寄存器的使用

用于组织存储器的主要数据结构是段。在 386、486 中，段是线性地址可变大小的块，并有一些属性与之相联系。有三种主要的类型的段：码段(存放程序代码)、数据段和堆栈段(堆栈

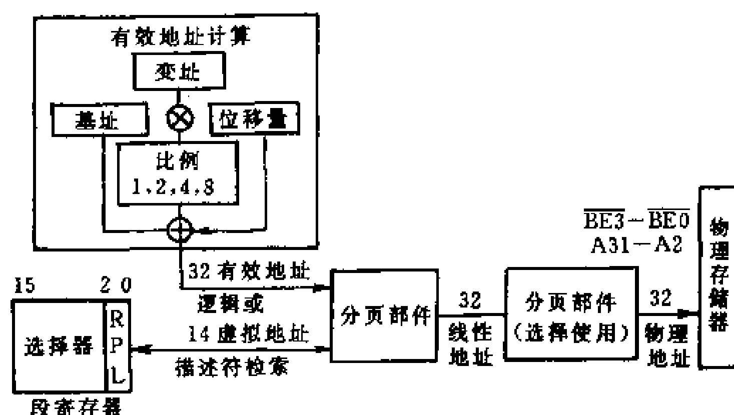


图 3-26　地址转换

操作所应用的段)。这些段的大小可变,可以小至一个字节或大至 4G 字节。

为了使指令的编码紧凑并提高处理器的性能,并不需要每条指令都明确指定哪一个段寄存器。系统有一种默认设置,如表 3-11 所示:

表 3-11　段寄存器选择规则

存储器访问类型	隐含使用的段	可能超越的段
取码	CS	无
PUSH,PUHA 指令的目的	SS	无
POP,POPA 指令的源	SS	无
使用具有有效地址的基址寄存器		
进行的其他数据访问:		
[EAX]	DS	CS,SS,ES,FS,GS
[EBX]	DS	CS,SS,ES,FS,GS
[ECX]	DS	CS,SS,ES,FS,GS
[EDX]	DS	CS,SS,ES,FS,GS
[ESI]	DS	CS,SS,ES,FS,GS
[EDI]*	DS	CS,SS,ES,FS,GS
[EBP]	DS	CS,SS,ES,FS,GS
[ESP]	DS	CS,SS,ES,FS,GS

* 对于 STOS 和 MOVS(以及 REP STOS 和 REP MOVS)指令的存储器目标地址所进行的数据访问,使用 DI 作为基址寄存器而 ES 作为段寄存器,没有超越的可能。

一般地说,数据参量使用 DS 寄存器中所包含的选择器。堆栈参量使用 SS 寄存器,同时以 ESP 寄存器作为偏移量。而指令读取使用 CS 寄存器,指令指针中的内容作为偏移量。对数据操作,可以用段超越前缀来使用别的段寄存器。

关于任何段的基地址的覆盖问题没有限制。所有 6 个段都可以将基地址设置为零并用 4G 字节线性地址空间建立一个系统。这样建立的系统其虚拟地址空间和线性地址空间是同样的。

3. I/O 空间

80386、80486 有两个不同的物理地址空间、存储器和 I/O。从一般的意义上说,80386、80486 也支持存储器对应的 I/O 方式,但 80386、80486 主要使用专用的端口寻址方式有单独

的 I/O 空间，外部设备放在 I/O 空间。

I/O 空间由 64K 字节组成，它可以分为 64K 个 8 位端口，32K 个 16 位端口或 16K 个 32 位端口，或加起来不超过 64 字节的任意端口的组合。这 64K 字节的 I/O 空间，相应于存储器的物理地址而不是线性地址，因为 I/O 指令不通过分段和分页部件。

M/$\overline{IO}$引出脚相当于一条附加的地址线，系统是用这条引线来区分是存储器空间还是 I/O 空间。

I/O 空间是通过 IN 和 OUT 指令来存取的，端口地址由 DL、DX 或 ESX 寄存器提供。当使用所有 8 位和 16 位端口地址时，地址线的高位部分扩展为零。I/O 指令驱动 M/$\overline{IO}$引脚为低。

I/O 端口地址中的 00F8H～00FFH 被保留为 INTEL 公司使用。数值协处理器也驻留在这个 I/O 空间，地址为 800000F8H～800000FCH。

第四节　86X86 系统的操作和总线周期

一个微型机系统为了完成自身的功能，需要 CPU 执行多种操作。以下几方面是 8086 的主要操作。

1. 系统的复位和启动操作；
2. 暂停操作；
3. 总线操作；
4. 中断操作；
5. 最小模式下的总线保持；
6. 最大模式下的总线请求/允许。

一、系统的复位和启动操作

8086/8088 的复位和启动操作是通过 RESET 引脚上的触发信号来执行的。

8086/8088 要求复位信号 RESET 起码维持 4 个时钟周期的高电平，如果是初次加电引起的复位，则要求维持不小于 50μS 的高电平。

当 RESET 信号一进入高电平时，8086/8088CPU 就会结束现行操作，并且，只要 RESET 信号停留在高电平状态，CPU 就维持在复位状态。在复位状态，CPU 各内部寄存器都被设为初值，如表 3-12 所示。

表 3-12　复位时各内部寄存器的值

标志寄存器	清零
指令指针(IP)	0000H
CS 寄存器	FFFFH
DS 寄存器	0000H
SS 寄存器	0000H
ES 寄存器	0000H
指令队列	空
其他寄存器	0000H

从表 3-12 中看到，在复位的时候，代码段寄存器 CS 和指令指针寄存器 IP 分别初始化为 FFFFH 和 0000H。所以，8086/8088 在复位之后再重新启动时，便从内存的 FFFF0H 处开始执行指令。因此，一般在 FFFF0H 处存放一条无条件转移指令，转移到系统程序的入口处。这样，系统一旦被启动，便自动进入系统程序。

在复位时，由于标志寄存器被清零，即所有标志位都被清除了，这样，所有从 INTR 引脚引入的可屏蔽中断都不能得到允许。因而，系统程序在适当时候，总是要通过指令（开放

中断指令 STI)来设置中断允许标志。

复位信号 RESET 从高电平到低电平的跳变会触发 CPU 内部的一个复位逻辑电路，经过 7 个时钟周期之后，CPU 就被启动而恢复正常工作，即从 FFFF0H 处开始执行程序。

在 RESET 信号电平变化时，信号线的输出有什么变化呢？

RESET 信号变为高电平后，再过 1 个时钟周期，所有三态输出线就被设置成高阻状态，并且一直维持高阻状态，直到 RESET 信号回到低电平。但在进入高阻状态的前半个时钟周期，也就是在前一个时钟周期的低电平期间(见图 3-27 所示)，这些三态输出线被设置成不作用状态。等到时钟信号又成为高电平时，三态输出线才进入高阻状态。

三态输出线包括 $AD_{15}\sim AD_0$、$A_{19}/S_6\sim A_{16}/S_3$、$\overline{BHE}/S_7$、$M/\overline{IO}$、$DT/\overline{R}$、$\overline{DEN}$、$\overline{WR}$、$\overline{RD}$和$\overline{INTA}$。还有几条非三态输出线，在复位之后会处于无效状态，但不浮空，它们是 ALE、HLDA、$\overline{RQ}/\overline{GT_0}$、$RQ/\overline{GT_1}$、$QS_0$、$QS_1$。

图 3-27 是复位操作的时序。表 3-13 表示了复位操作时，8086 的总线信号。

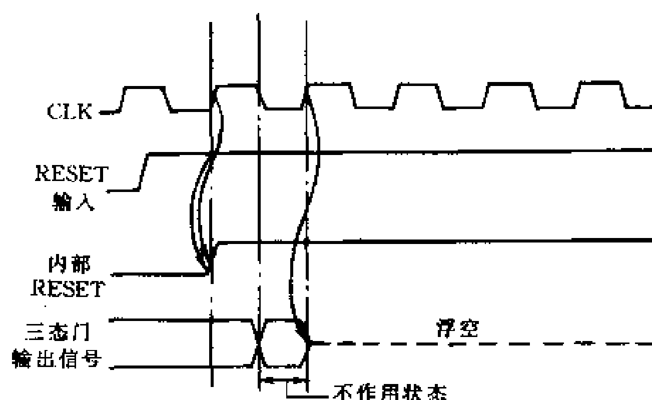


图 3-27　8086 的复位时序

表 3-13　复位时 8086 的总线信号

信　　号	状　　态
$AD_{15}\sim AD_0$	先置成不作用状态，再进入三态。不作用状态占进入三态前的半个时钟周期(即时钟为低电平期间)
$A_{19}/S_6\sim A_{16}/S_3$	
$\overline{BHE}/S_7$	
$\overline{S_2}(M/\overline{IO})$	
$\overline{S_1}(DT/\overline{R})$	
$\overline{S_0}(\overline{DEN})$	
$LOCK(\overline{WR})$	
$\overline{RD}$	
$\overline{INTA}$	
ALE	低
HLDA	低
$\overline{RQ}/\overline{GT_0}$	高
$\overline{RQ}/\overline{GT_1}$	高
QS_0	低
QS_1	低

二、总线操作

8086/8088CPU 为了要与存储器或外设端口交换数据，需要执行一个总线周期，这就是总线操作。

按照数据传输方向来分，总线操作可以分为读总线操作和写总线操作。读总线操作就是指 CPU 从存储器或外设端口读取数据；写总线操作是指 CPU 将数据写入存储器或外设端口。

1. 总线周期的概念

在 8086 中，一个最基本的总线周期由 4 个时钟周期组成，时钟周期是 CPU 的基本时间计量单位，由主频决定。例如 8086 的主频为 5MHz，1 个时钟周期就是 200 ns。8086－1 的主频为 10MHz，则 1 个时钟周期就是 100 ns。一个时钟周期又称为一个状态 T，因此基本总线周期用 T_1、T_2、T_3、T_4 表示。图 3-28 示出典型的 BIU 总线周期波形图。在 T_1，CPU 把读/写的存储单元或 I/O 端口的地址放到总线上；在 T_2～T_4，若是写总线周期，CPU 从 T_2 起到 T_4，把数据送到总线上；若是"读"总线周期，CPU 则从 T_3 起到 T_4 期间从总线上接收数据，T_2 时，总线浮空，允许 CPU 有个缓冲时间把输出地址的写方式转换成输入数据的读方式。可见，AD_0～AD_{15}和 A_{19}/S_6～A_{16}/S_3 在总线周期的不同状态传送不同的信号，这就是 8086 采用的分时复用总线。

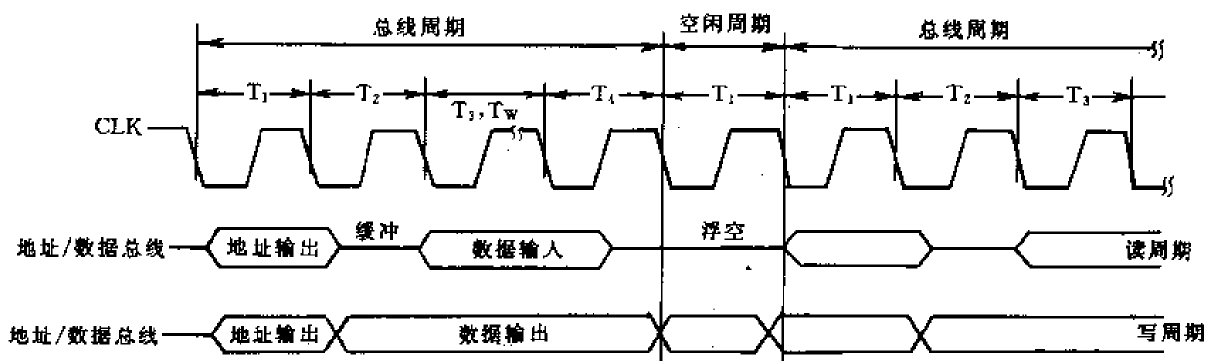


图 3-28　典型的 BIU 总线周期波形图

需要指出：

1）在有些情况下，被写入数据或被读取数据的存储器或外设在速度上跟不上 CPU 的要求时，就会由存储器或外设通过 READY 信号线在 T_3 状态启动之前向 CPU 发一个 READY 无效的信号，表示未准备就绪，于是 CPU 将在 T_3 之后插入 1 个或多个附加的时钟周期 T_W，T_W 又叫等待状态。在 T_W 状态，总线上的信息情况维持 T_3 状态的信息情况。当存储器或外设完成数据的读/写准备时，便在 READY 线上发出有效信号，CPU 接到此信号，会自动脱离 T_W 而进入 T_4 状态。

2）总线周期只用于 CPU 和存储器或 I/O 端口之间传送数据和供填充指令队列，如果在 1 个总线周期之后，不立即执行下 1 个总线周期，那么，系统总线就处于空闲状态，即执行空闲周期 T。在空闲周期中，可以包含一个时钟周期或多个时钟周期，这期间，在高 4 位的总线上，CPU 仍然驱动前一个总线周期的状态信息；而在低 16 位的总线上，则视前一个总线周期是写还是读周期来确定。若前一个总线周期为写周期，CPU 会在总线的低 16 位继续去驱动数据信息；若前一个总线周期为读周期，CPU 则使总线的低 16 位处于浮空状态。

在空闲周期中，尽管CPU对总线进行空操作，但在CPU内部，仍然进行着有效的操作。比如执行某个运算，在内部寄存器之间传输数据等等。按照8086/8088编程结构，可以知道这些操作都是由执行部件进行的。实际上，总线空操作是总线接口部件对执行部件的等待。

下面，以8086为例，讲述总线操作的典型时序关系和具体操作过程。

2. 最小方式下的读总线操作

图3-29表示CPU从存储器或外设端口读取数据的时序。

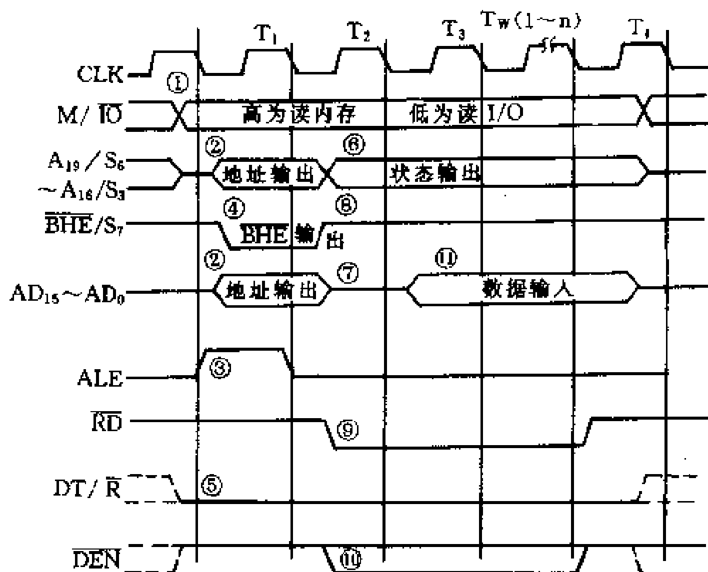


图3-29　8086读周期的时序

一个最基本的读周期包含4个状态，即T_1、T_2、T_3、T_4。在存储器和外设速度较慢时，要在T_3之后插入1个或几个等待状态T_W。

(1)T_1状态

为了从存储器或I/O端口读出数据，首先要用M/$\overline{IO}$信号指出CPU是从内存还是从I/O端口读，所以，M/$\overline{IO}$信号在T_1状态成为有效(见图3-29①)。如果是从存储器读数据，则M/$\overline{IO}$为高；如果是从I/O端口读数据，则M/$\overline{IO}$为低。M/$\overline{IO}$信号的有效电平一直保持到整个总线周期的结束，即T_4状态。

此外，CPU要指出所读取的存储单元或I/O端口的地址。8086的20位地址信号是通过多路复用总线输出的，高4位地址通过地址状态线A_{19}/S_6～A_{16}/S_3送出，低16位地址通过地址/数据线AD_{15}～AD_0送出。在T_1状态的开始，20位地址信息就通过这些引脚送到存储器和I/O端口(见图3-29②)。

地址信息必须被锁存起来，这样才能在总线周期的其他状态，往这些引脚上传输数据和状态信息。为了实现对地址的锁存，CPU便在T_1状态从ALE引脚上输出一个正脉冲作为地址锁存信号(见图3-29③)。在ALE的下降沿到来之前，M/$\overline{IO}$信号、地址信号均已有效。锁存器8282正是用ALE的下降沿对地址进行锁存。

$\overline{BHE}$信号也在T_1状态通过$\overline{BHE}/S_7$引脚送出(见图3-29④)，它用来表示高8位数据总线上的信息可以使用。$\overline{BHE}$信号常常作为奇地址存储体的体选信号，配合地址信号来实现存储单元的寻址，因为奇地址存储体中的信息总是通过高8位数据线来传输。顺便提一句，偶地址

存储体的体选信号就是最低地址 A_0。

除此以外，当系统中接有数据总线收发器时，要用到 $DT/\overline{R}$ 和 $\overline{DEN}$ 作为控制信号。前者作为对数据传输方向的控制，后者实现数据的选通。为此，在 T_1 状态，$DT/\overline{R}$ 端输出低电平，表示本总线周期为读周期，即让数据总线收发器接收数据(见图 3-29⑤)。

(2) T_2 状态

在 T_2 状态，地址信号消失(见图 3-29⑦)，此时，AD_{15}～AD_0 进入高阻状态，以便为读入数据作准备；而 A_{19}/S_6～A_{16}/S_3 及 $\overline{BHE}/S_7$ 引脚上输出状态信息 S_7～S_3(见图 3-29⑥⑧)。不过，在当前 CPU 设计中，S_7 未被赋予任何意义。

$\overline{DEN}$ 信号在 T_2 状态变为低电平(见图 3-29⑩)，从而在系统中接有总线收发器时，获得数据允许信号。

在 T_2 状态，CPU 在 $\overline{RD}$ 引脚上输出信号，$\overline{RD}$ 信号送到系统中所有的存储器和 I/O 接口。但是，只有被地址信号选中的存储单元或 I/O 端口，才会被 $\overline{RD}$ 信号从中读出数据，而将数据送到系统的数据总线上。

(3) T_3 状态

在基本总线周期的 T_3 状态，内存单元或 I/O 端口将数据送到数据总线上。CPU 通过 AD_{15}～AD_0 接收数据。

(4) T_W 状态

当系统中所用的存储器或外设的工作速度较慢，从而不能用最基本的总线周期执行读操作时，系统中就要用一个电路来产生 READY 信号，READY 信号通过时钟发生器 8284A 传递给 CPU。CPU 在 T_3 状态的前沿(下降沿处)对 READY 信号进行采样。如果 CPU 没有在 T_3 状态的一开始采样到 READY 信号(当然，在这种情况下，在 T_3 状态，数据总线上不会有数据)为低电平，那么，就会在 T_3 和 T_4 之间插入等待状态 T_W，T_W 可以为 1 个，也可以为多个。以后，CPU 在每个 T_W 的前沿对 READY 信号进行采样，直到 CPU 接收到高电平的 READY 信号后，再把当前 T_W 状态执行完，便脱离 T_W 而进入 T_4。

在最后一个 T_W 状态，数据肯定已经出现在数据总线上。所以，最后一个 T_W 状态中总线的动作和基本总线周期中 T_3 状态的完全一样。而在其它的 T_W 状态，所有控制信号的电平和 T_2 状态一样，但数据信号尚未出现在数据总线上。

(5) T_4 状态

在 T_4 状态和前一个状态交界的下降沿处，CPU 对数据总线进行采样，从而获得数据。

3. 最小方式下的写总线操作

图 3-30 表示了 8086CPU 往存储器或外设端口写入数据的时序。

和读操作一样，最基本的写操作周期也包含 4 个状态，即 T_1、T_2、T_3、和 T_4。当存储器和外设速度较慢时，在 T_3 和 T_4 状态之间，CPU 会插入 1 个或几个等待状态 T_W。

下面，对写总线周期中各状态，CPU 的输出信号情况作一个具体说明。

(1) T_1 状态

在 T_1 状态，CPU 要用 $M/\overline{IO}$ 信号指出当前执行的写操作是将数据写入内存还是写入 I/O 端口。如果是写入内存，则 $M/\overline{IO}$ 为高电平，如果是写入 I/O 端口，则 $M/\overline{IO}$ 为低电平。所以，在 T_1 状态，$M/\overline{IO}$ 便进入有效电平(见图 3-30①)，有效电平一直保持到 T_4 状态才结束。

CPU 在 T_1 状态还提供了地址信号来指出具体要往哪一个存储单元或 I/O 端口写入数据

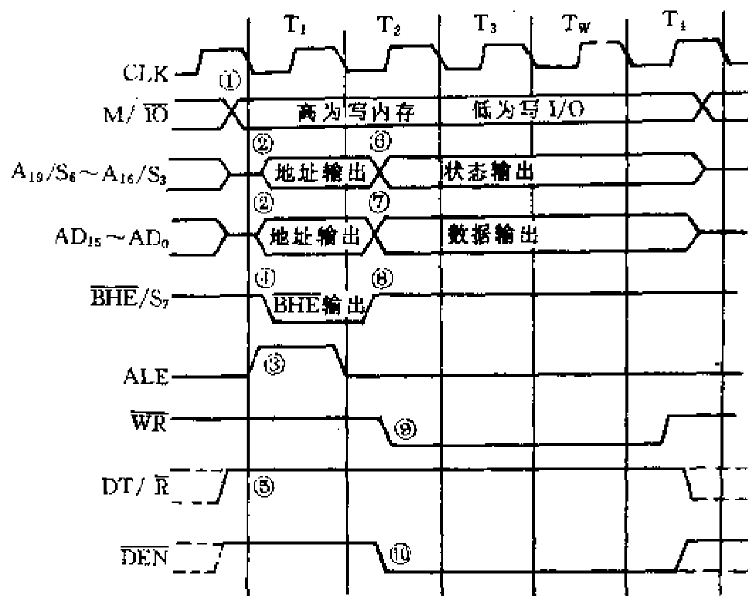


图 3-30　8086 写周期的时序

（见图 3-30②）。

高 4 位地址和状态信号是从同一组引脚上分时送出的，低 16 位地址和数据是从同一组引脚上分时传输的，所以，必须把地址信息锁存起来。为了实现地址的锁存，CPU 在 T_1 状态从 ALE 引脚上送出一个地址锁存信号。地址锁存信号是一个正向脉冲（见图 3-30③），在 ALE 的下降沿到来之前，地址信号和 $\overline{BHE}$、M/$\overline{IO}$都已经有效，地址锁存器 8282 就是利用 ALE 的下降沿对地址信号和$\overline{BHE}$信号进行锁存的。

$\overline{BHE}$信号是数据总线高位有效信号，CPU 也是在 T_1 状态的开始就使$\overline{BHE}$信号有效（见图 3-30④）。$\overline{BHE}$信号在实际系统中作为存储体的体选信号，配合地址信号来实现对奇地址存储体中存储单元的寻址。偶地址存储体的体选信号一般用 A_0。当地址最低位 A_0 为 0 时，选中偶地址存储体。

当系统中有数据收发器时，在写总线周期中，要用$\overline{DEN}$信号作为数据收发器的允许信号，而用 DT/$\overline{R}$ 信号来控制收发器的数据传输方向。为此，CPU 在 T_1 状态就使 DT/$\overline{R}$ 信号成为高电平（见图 3-30⑤），以表示本总线周期执行写操作。

(2) T_2 状态

地址信号发出之后，CPU 立即从地址/数据复用引脚 $AD_{15} \sim AD_0$ 发出要写到存储单元或 I/O 端口的数据（见图 3-30⑦），数据信息会一直保持到 T_4 状态的中间。与此同时，CPU 在 $A_{19}/S_6 \sim A_{16}/S_3$ 引脚上发出状态信号 $S_6 \sim S_3$（见图 3-30⑥），而$\overline{BHE}$信号则消失（见图 3-30⑧）。

在 T_2 状态，CPU 从$\overline{WR}$引脚上发出信号$\overline{WR}$，写信号与读信号一样，一直维持到 T_4 状态（见图 3-30⑨）。在实际系统中，写信号送到所有的存储器（只读存储器除外）和 I/O 接口，但是，只有被地址信号选中的存储单元或 I/O 端口，才被$\overline{WR}$信号写入数据。

(3) T_3 状态

在 T_3 状态，CPU 继续提供状态信息和数据，并且继续维持$\overline{WR}$、M/$\overline{IO}$及$\overline{DEN}$信号为有效电平。

(4) T_W 状态

如果系统中设置 READY 电路，并且 CPU 在 T_3 状态的一开始未收到“准备好”信号，那么，会在状态 T_3 和 T_4 之间插入 1 个或几个等待周期。直到在某个 T_W 的前沿处，CPU 采样到“准备好”信号后，便将 T_W 状态作为最后一个等待状态。执行完 T_W 状态后进入 T_4 状态。在 T_W 状态，总线上所有控制信号的情况和 T_3 时一样，数据总线上也仍然保持要写入的数据。

(5) T_4 状态

在 T_4 状态，CPU 认为存储器或外设端口已经完成数据的写入，因而，数据从数据总线上被撤除，各控制信号线和状态信号线也进入无效状态。此时，$\overline{DEN}$信号总是进入高电平，从而使总线收发器不工作。

4. 最大模式下的读总线操作

最大模式下，8086/8088 的读总线操作在逻辑上是和最小模式下的读操作一样的，但在时序上分析时，最大模式下要考虑 CPU 和总线控制器两者产生的信号。

CPU 在最大模式下仍然提供$\overline{RD}$信号，此外，总线控制器还由 S_2，S_1，S_0 状态信号产生存储器读信号$\overline{MRDC}$和输入/输出端口读信号$\overline{IORC}$。

和$\overline{RD}$信号相比，$\overline{MRDC}$和$\overline{IORC}$信号除了指出具体的读取目标到底是存储还是输入/输出端口外，信号的交流特性也要比$\overline{RD}$好得多，所以，尽管从时间上看，$\overline{MRDC}$、$\overline{IORC}$和$\overline{RD}$都是在总线周期的 T_2 状态发出，但在最大模式中，总是尽量利用总线控制器输出的读信号$\overline{MRDC}$和$\overline{IORC}$.

如果用$\overline{RD}$信号，那么，最大模式中的读操作的时序和最小模式中的一样。所以下面只考虑用 8288 输出的读信号来执行读操作的时序。

在每个总线周期的开始之前一段时间，$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$必定被置为高电平。总线控制器只要一检测到$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$中任何 1 个或几个从高电平状态开始有变化，便立即开始一个新的总线周期。如果是总线读周期，则各信号之间的时序关如图 3-31 所示，图中带星号的信号是由总线控制器发出的。

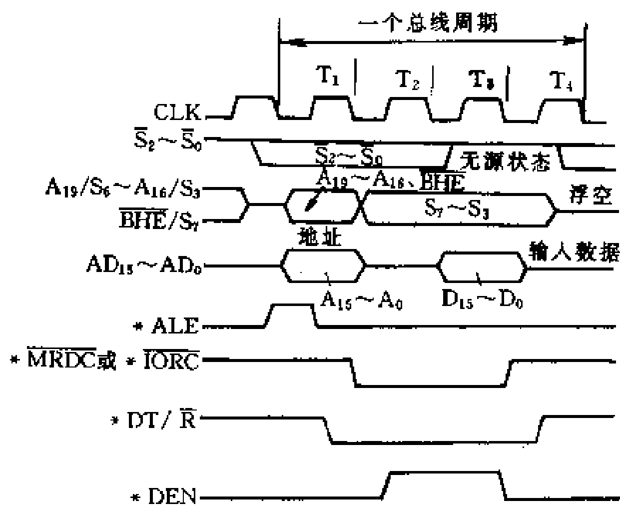


图 3-31 最大模式下的读操作时序

从图 3-31 中可以看到：

(1) 在 T_1 状态，CPU 将地址的低 16 位通过 $AD_{15}\sim AD_0$ 发出，地址的高 4 位通过 $A_{19}/S_6\sim A_{16}/S_3$发出，总线控制器从 ALE 引脚上输出一个正向的地址锁存脉冲，系统中的地址锁存器利用这一脉冲将地址锁存起来。此外，总线控制器还为总线收发器提供数据传输方向控制信号 $DT/\overline{R}$。在 T_1 状态，$DT/\overline{R}$ 进入低电平，表示当前总线周期执行读操作，此低电平一直维持到 T_4 为止。

(2) 在 T_2 状态，CPU 输出状态信号 $S_7\sim S_3$。总线控制器在 T_2 状态的时钟上升沿处，使$\overline{DEN}$信号有效，于是，总线收发器允许。总线控制器还根据$\overline{S_2}$、$\overline{S_1}$和$\overline{S_0}$的电平组合发出读信号$\overline{MRDC}$或者$\overline{IORC}$，送到存储器或者输入/输出端口，去执行存储器读操作或者输入/输出端口读操作。$\overline{MRDC}$或$\overline{IORC}$在 T_2 状态进入低电平后，将一直维持此有效电平到 T_4。此外，CPU 将地址/数据引脚设置为高阻状态，以便为输入数据作好准备。

(3) 在 T_3 状态，如果所读取的存储器或者外设速度足够快，则此时，它们已经把数据送到数据总线上，于是 CPU 就可以获得数据。$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$全部进入高电平，也就是进入无源状态，总线的无源状态从 T_3 一直维持到 T_4。一旦进入无源状态，就意味着很快可以启动一个新的总线

周期。

(4) 在 T_4 状态，数据从总线上消失，状态信号引脚 $S_7 \sim S_3$ 进入高阻状态，而$\overline{S_2}$,$\overline{S_1}$,$\overline{S_0}$则按照下一个总线周期的操作类型产生电平变化。

和最小模式下的总线读操作类似，如果存储器和外设的速度比较慢，则需要使用 READY 信号来联络。如果在 T_3 状态开始时，READY 仍没有达到高电平，则在 T_3 状态和 T_4 状态之间插入 T_W 状态，T_W 状态可为 1 个或几个。

5. 最大模式下的写总线操作

在 8086/8088 按最大式工作时，CPU 通过总线控制器为存储器和输入/端口提供两组写信号：一组是普通的存储器写信号$\overline{MWTC}$和普通的输入/输出端口写信号$\overline{IOWC}$；另一组是提前的存储器写信号$\overline{AMWC}$和提前的输入/输出端口写信号$\overline{AIOWC}$。提前的写信号比普通写信号提前整整一个时钟周期开始起作用。

图 3-32 是最大模式下写总线操作的时序图。图中带星号的信号由总线控制器提供。

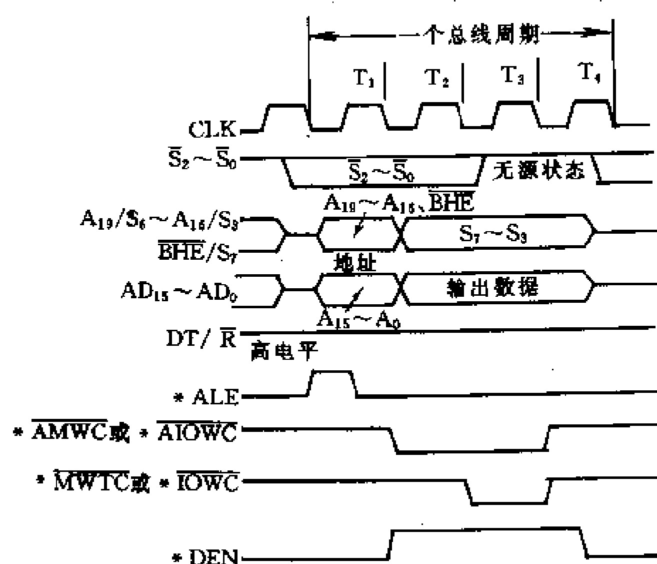


图 3-32 最大模式下的写操作时序

从图 3-32 中见到，和读操作一样，在写操作总线周期开始之前，S_2、S_1、S_0 就已经按照操作类型设置好相应的电平。从 T_1 状态开始，整个总线周期的时序关系如下：

(1) 在 T_1 状态，CPU 从 $AD_{15} \sim AD_0$ 引脚上输出要写入存储器单元或输入/输出端口的低 16 位，而从 $A_{19}/S_6 \sim A_{16}/S_3$ 引脚输出地址的高 4 位。CPU 还使数据总线高 8 位有效信号$\overline{BHE}$进入有效电平。在 T_1 状态，总线收发器使 $DT/\overline{R}$ 输出高电平，以表示本总线周期进行写操作，数据总线收发器根据 $DT/\overline{R}$ 为高电平决定了数据传输方向为发送方向。

(2) 在 T_2 状态，总线控制器使$\overline{DEN}$输出高电平，于是数据总线收发器得到允许。在$\overline{DEN}$输出高电平的同时，提前的存储器写信号$\overline{AMWC}$或者提前的输入/端口写信号$\overline{AIOWC}$也为低电平即有效电平，并且一直维持到 T_4。在总线写周期中，CPU 从 T_2 状态开始就把数据送到数据总线上。

(3) 在 T_3 状态，总线控制器使普通的存储器写信号$\overline{MWTC}$或者普通的输入/输出端口写信号$\overline{IOWC}$成为低电平，并且一直维持到 T_4。由此可见，两个提前的写信号$\overline{AMWC}$和$\overline{AIOWC}$比普通的写信号$\overline{MWTC}$和$\overline{IOWC}$超前了整整一个时钟周期，这样，一些较慢的设备或者存储器芯片就可以得到一个额外的时钟周期执行写操作。和总线读操作一样，在总线写周期的 T_3 状态，S_2、S_1、S_0 全部进入高电平，于是，总线进入无源状态，从而为启动下一个总线周期作好了准备。

(4) 在 T_4 状态，CPU 将地址/数据引脚 $AD_{15} \sim AD_0$ 以及地址/状态引脚 $A_{19}/S_6 \sim A_{16}/S_3$ 设置为高阻状态。写信号$\overline{AMWC}$、$\overline{MWTC}$或者$\overline{AIOWC}$、$\overline{IOWC}$都被撤消。而且，数据总线收发器

信号$\overline{DEN}$也进入低电平，这样，便使数据总线收发器停止工作。$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$则按照下一个总线周期的操作类型产生变化，从而启动一个新的总线周期。

在最大模式下的总线写周期中，如果用了 READY 信号，并且当 READY 信号没赶在 T_3 状态开始之前来到时，也会在 T_3 状态和 T_4 状态之间插入 1 个或几个等待状态。

三、最小方式下的总线保持

当一个系统中具有多个总线主模块时，CPU 以外的其他总线主模块为了获得对总线的控制权，需要向 CPU 发出使用总线的请求信号。而 CPU 得到请求之后，如果同意让出总线，就要向请求使用总线的主模块发应答信号。8086/8088 为此提供了一对专用于最小模式下的总线控制联络信号 HOLD 和 HLDA。

HOLD 叫总线保持信号，是由其它主模块发给 CPU 的。当某个总线主模块企图使用系统总线时，便发出高电平的 HOLD 信号，然后等待 CPU 发来总线保持回答信号。

在每个时钟脉冲的上升沿处，CPU 会对 HOLD 引脚上的信号进行检测。如果检测到 HOLD 处于高电平状态，并且允许让出总线周期的 T_4 状态或者空闲状态 T_1 之后的下一个时钟周期，CPU 会发出 HLDA 信号，从而 CPU 便将总线让给发出总线保持请求的设备，直到此后这个发出总线保持请求的设备又将 HOLD 信号变为低电平，CPU 才又收回总线控制权。

8086/8088 CPU 一旦让出总线控制权，便使地址/数据引脚、地址/状态引脚以及控制信号引脚$\overline{RD}$、$\overline{WR}$、$\overline{INTA}$、M/$\overline{IO}$、$\overline{DEN}$及 DT/$\overline{R}$ 都处于浮空状态，这样，CPU 和数据总线、地址总线以及上

述控制信号之间就暂时没有关系了。不过，ALE 信号引脚不悬空。

图 3-33 是表示总线保持请求和保持响应操作的时序图。

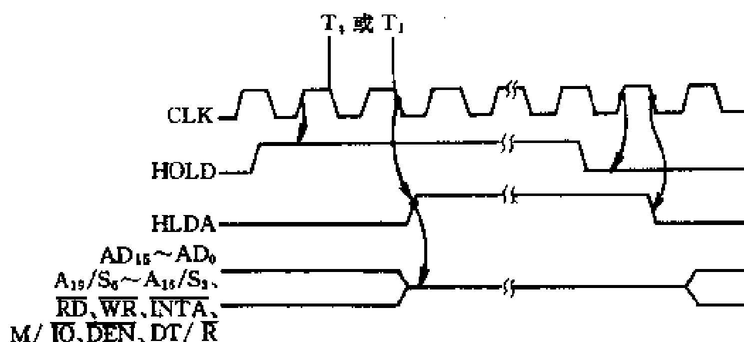


图 3-33 总线保持请求/保持响应时序

在考虑总线保持请求和保持响应操作过程时，有三个问题需要注意：

1. 外部的总线主模块将 HOLD 信号变为高电平后，CPU 要在下一个时钟周期的上升沿时才能检测到，随后 CPU 在此时钟周期（假定正好此时钟周期为 T_4 或者 T_1）的下降沿处将 HLDA 信号升为高电平。CPU 在测得 HOLD 信号时，假如并不是处在 T_4 或者 T_1 状态，那么可能会延迟几个时钟周期，等到 T_4 状态或者 T_1 状态时才发出 HLDA 信号。

2. HOLD 请求信号直接影响 8086/8088 总线接口部件（BIU）的工作，但是对执行部件（EU）的影响只是间接的。当 CPU 使 HLDA 升为高电平时，尽管 CPU 所有的三态引脚进入悬空状态，但是 EU 将继续执行已进入指令队列中的指令，直到遇见一条需要使用总线的指令时，执行部件才不得不停下来。当然，指令队列中的指令执行完后，执行部件也会停下来。由此可见，CPU 和其他获得总线控制权的主模块之间在操作上有一段小小的重叠。

3. 当发总线保持请求的设备将 HOLD 信号降为低电平时，CPU 也将 HLDA 信号降为低电平，但是，CPU 不会马上重新驱动地址总线、数据总数和控制线，而是使这些引脚继续悬空，

直到 CPU 需要执行一个总线操作时，才结束这些引脚的浮空状态。这样以来，就可能出现这样的情况，在某一小段时间中，没有任何一个模块在驱动总线。为了防止在总线控制的切换期间，由于没有任何主模块的驱动而造成控制线电平飘移到最小电平以下，所以，在控制线和电源之间需要连接一个提拉电阻。

四、最大方式下的总线请求/允许

在最大模式下，8086/8088 也提供了总线主模块之间传递总线控制权的手段，CPU 以外的其他总线主模块包括协处理器、DMA 控制器。它们可以连接在局部总线上，去共享 CPU 和系统总线之间的接口，即共享地址锁存器、数据总线收发器、总线控制器，当然也共享系统总线。与最小模式下不同的是，在最大模式下，总线控制信号不再是 HOLD 和 HLDA，而是将这两个信号引脚变成功能更加完善的两个双向信号引脚$\overline{RQ}/\overline{GT_0}$和$\overline{RQ}/\overline{GT_1}$，它们都称为总线请求/总线允许信号端，可以分别连接两个不同的总线主模块。

$\overline{RQ}/\overline{GT_0}$和$\overline{RQ}/\overline{GT_1}$有着完全相同的功能，但是$\overline{RQ}/\overline{GT_0}$比$\overline{RQ}/\overline{GT_1}$的优先级要高。也就是说，如果$\overline{RQ}/\overline{GT_0}$和$\overline{RQ}/\overline{GT_1}$都连接了其它总线主模块，当两条引脚上同时出现总线请求时，CPU 会对$\overline{RQ}/\overline{GT_0}$先发出允许信号，等到 CPU 再次得到总线控制权时，才响应$\overline{RQ}/\overline{GT_1}$引脚上的请求。不过，如果 CPU 已经把总线控制权让给连接$\overline{RQ}/\overline{GT_1}$的主模块，此时又在$\overline{RQ}/\overline{GT_0}$引脚上收到另一个主模块的总线请求，这时，要等前一个主模块释放总线以后，CPU 收回了总线控制权，才会去响应 RQ/GT0 引脚上的总线请求。可见 CPU 对总线请求的处理并不是像对中断请求的处理那样可以允许“嵌套”。

图 3-34 表示了 8086、8087 和 8089 通过$\overline{RQ}/\overline{GT_0}$、$\overline{RQ}/\overline{GT_1}$的连接关系。

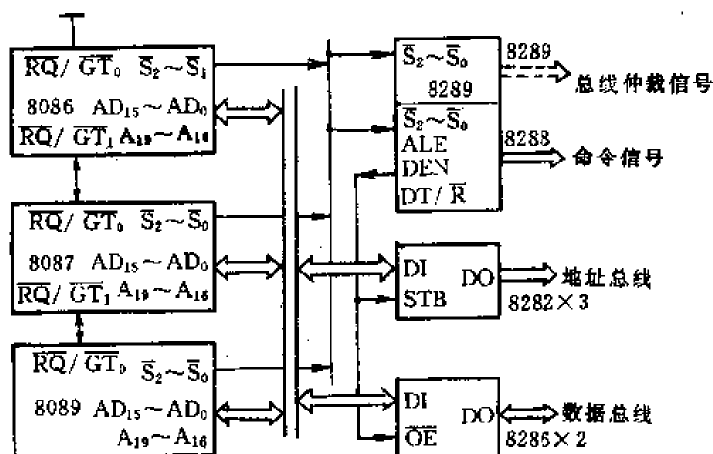


图 3-34　8086 通过$\overline{RQ}/\overline{GT}$和协处理器的连接

图 3-35 表示了最大模式中，CPU 以外的其他总线主模块请求总线使用权和 8086/8088 允许其他主模块获得总线使用权的时序。

对最大模式下的总线请求/允许/释放时序，我们作如下说明：

1. 当 CPU 以外的一个总线主模块要求使用系统总线时，从$\overline{RQ}/\overline{GT}$(即$\overline{RQ}/\overline{GT_0}$或者$\overline{RQ}/\overline{GT_1}$，以下类同)线上往 CPU 发一个负脉冲，宽度为一个时钟周期。

2. CPU 在每个时钟周期的上升沿处对$\overline{RQ}/\overline{GT}$引脚进行检测，如果测得外部来了一个负脉冲，则在下一个 T_4 状态或 T_1 状态从同一引脚$\overline{RQ}/\overline{GT}$上往请求总线使用权的主模块发一个允许脉冲，这个允许脉冲的宽度也相当于一个时钟周期。不过要注意，总线请求负脉冲是外部发给 CPU 的，而总线允许脉冲的传输方向正好相反，它是由 CPU 发给其他外部主模块的。CPU 一旦发出了响应脉冲，各地址/数据引脚、地址/状态引脚以及$\overline{RD}$、$\overline{LOCK}$、$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$、$\overline{BHE}/S_7$便处于高阻状态，于是在逻辑上暂时和总线断开。

3. 协处理器或其他总线主模块收到 CPU 发出的允许脉冲后，便得到了总线控制权，于是它可以对总线占用一个或几个总线周期。

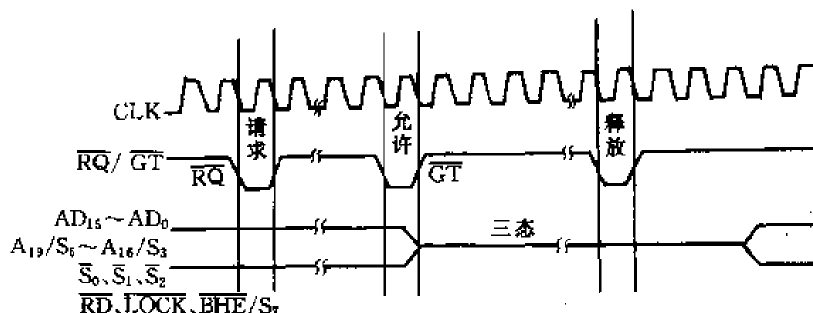


图 3-35 最大模式下的总线请求/允许/释放时序

4. 当外部主模块准备释放总线时，便在$\overline{RQ}/\overline{GT}$线上往 CPU 发一个释放脉冲，释放脉冲的宽度也为一个时钟周期。CPU 检测到释放脉冲后，在下一个时钟周期便收回了总线控制权。

从时序说明中，可以见到，每次总线控制权的切换都是通过三个环节来实现的，即协处理器或其它主模块发请求脉冲，CPU 发允许脉冲，外部主模块用完总线后发释放脉冲。这些脉冲都是负脉冲，但并不在一个方向上传输。

另外，在每次总线控制权改变后，总线上必须有一个空闲周期。即在一个时钟周期中，没有任何一个主模块使用总线。

实际上，在外部总线主模块发出总线请求时，还有一些附带条件对 CPU 响应总线请求作出限制，这就是：

① 如果 CPU 正在访问存储器或外设端口，那么，总线请求脉冲必须在 T_2 之前到达才有效，否则，外部主模块必须重发请求脉冲。

② 如果 CPU 正在用低 8 位数据线传送数据，则总线请求脉冲将无效。等到数据传送结束，外部主模块重发请求脉冲时，CPU 才予以响应。

③ 如果 CPU 正在执行中断响应的第一个总线周期，总线请求无效。

④ 如果 CPU 正在执行总线封锁指令，则总线请求无效。

所以，只有在总线空闲时遇到总线请求的情况下，CPU 才会在下一个时钟周期发出允许信号。

与最小模式下执行总线保持请求/保持响应操作时的情况一样，8086/8088 通过$\overline{RQ}/\overline{GT}$线发出响应负脉冲，并且释放总线后，CPU 仍然可以执行已经进入指令队列中的指令，直到需要请求一个总线周期为止。另外 CPU 收到其他主模块发出的释放脉冲后，也并不是立即驱动总线。

在 CPU 内部，$\overline{RQ}/\overline{GT_0}$和$\overline{RQ}/\overline{GT_1}$都配置了提拉电阻与电源相连，如果系统中未用这两个引脚，便可悬空。

其他操作在以后相应章节叙述。

五、80X86 系统时序介绍

80286 与 8086 时分总线不同，它们的地址总线与数据总线相互独立。在 80286CPU 中，对存储器或 I/O 进行读写操作的时间亦称为总线周期。在一个总线周期中，当有效数据置于数据总线上时，下一个总线的地址已开始输出到地址总线上，这就是流水作业的总线周期方式。在 80286/80386 中，基本总线周期由 T_S(Send Status)和 T_C(Perform Command) 两个时钟周期组成，而 8086 CPU 基本总线周期由 $T_1\sim T_4$ 个时钟周期组成。由此得出，80286/80386 最快读

周期只需两个时钟周期。80486 的一些基本指令不再由原来的微码控制，而是改由硬件直接执行，这样使读/写指令可以在单一时钟周期内完成。而 80586 CPU 则在每个时钟周期里可以执行 3 条指令。显然，80586 速度快得多。

思考题与习题

1. 8086/8088 CPU 的地址总线有多少位？其寻址范围是多少？

2. 8086/8088 CPU 分为哪两个部分？各部分主要由什么组成？

3. 什么叫队列？8086/8088 CPU 中指令队列有什么作用？其长度分别是多少字节？

4. 8086/8088 CPU 中有几个通用寄存器？有几个变址寄存器？有几个指针寄存器？通常哪几个寄存器亦可作为地址寄存器使用？

5. 8086/8088 CPU 中有哪些标志位？它们的含义和作用如何？

6. 对于由 8086/8088 CPU 组成的系统，堆栈的位置如何确立？试指出堆栈的首址(即 SS 中的值)是不是栈底？为什么？

7. INTEL 8086 与 8088 有何区别？

8. 在 8086/8088CPU 工作在最小模式时，

(1) 当 CPU 访问存储器时，要利用哪些信号？

(2) 当 CPU 访问外设接口时，要利用哪些信号？

(3) 当 HOLD 有效并得到响应时，CPU 的哪些信号置高阻？

9. 当 8086/8088 CPU 工作在最大模式时，

(1) $\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$可以表示 CPU 的哪些状态？

(2) CPU 的$\overline{RQ}$/$\overline{GT}$信号的作用？

10. 试求出下列运算后的各个状态标志，并说明进位标志和溢出标志的区别？

(1) 1278H+3469H

(2) 54E3H−27A0H

(3) 3881H+3597H

(4) 01E3H−01E3H

11. 8086 构成系统分为哪两个存储体？它们如何与地址、数据总线连接？

12. 什么是逻辑地址？什么是物理地址？它们之间有什么联系？各用在何处？

13. 什么是段基值？位移量？它们之间有何联系？

14. 若 CS 为 A000H，试说明现行代码段可寻址存储空间的范围？

15. 设现行数据段位于存储器 B0000H 到 BFFFFH 存储单元，DS 段寄存器内容为多少？

16. 8086/8088 CPU 使用的存储器为什么要分段？怎么分段？

17. 已知当前段寄存器的基值(DS)=021FH，(ES)=0A32H，(CS)=234EH，则上述各段在存储器空间中物理地址的首址及末地址号是什么？

18. 若(CS)=234EH 时，已知物理转移地址为 25432H. 问若(CS)的内容被指定成 1A31H 时，物理转移地址应为什么地址号？

19. 若(CS)=5200H 时，物理转移地址为 5B230H，则当 CS 的内容被设定为 7800H，物理转移地址应为多少？

20. 若 8086 工作于最小方式，试指出当 CPU 完成将 AH 中的内容送到物理地址为 9100H 的存储单元操作时，以下哪些引脚信号应为低电平：$\overline{BHE}/S_7$(总线周期的第一部分时间

时)、$\overline{RD}$、$\overline{WR}$、$M/\overline{IO}$、$DT/\overline{R}$。

21. 若 8086 工作于最大方式，试指出当 CPU 完成将 CL 内容传送到物理地址为 91003H 单元的操作时，8288 输出的总线命令信号哪些应变为有效(低电平)？

22. 8086 CPU 工作在最小模式(单 CPU)和最大模式(多 CPU)主要特点是什么？有何区别？

23. 某系统中已知当前(SS)=2580H，(SP)=0800H，请说明该堆栈段在存储器中的物理地址范围.若已知当前堆栈中已存有 10 个字节数据，那么 SP 内容应为什么值？

24. 8086 CPU 读/写总线周期各包含多少个时钟周期？什么情况下需要插入 T_W 等待周期？应插入多少个 T_W，取决于什么因素？什么情况下会出现空闲状态 T_I？

25. 现有 6 个字节的数据分别为 11H，22H，33H，44H，55H，66H，已知它们在存储器中的物理地址为 400A5H～400AAH. 若当前(DS)=4002H，请说明它们的偏移地址值。如果要从存储器中读出这些数据，需要访问几次存储器，各读出哪些数据？

26. 已知当前数据段中存有如图 3-36 所示的数据，现要求将最后两个字节改成 0DH，0AH，请说明需给出的段基值和偏移地址值，并说明其写入过程。

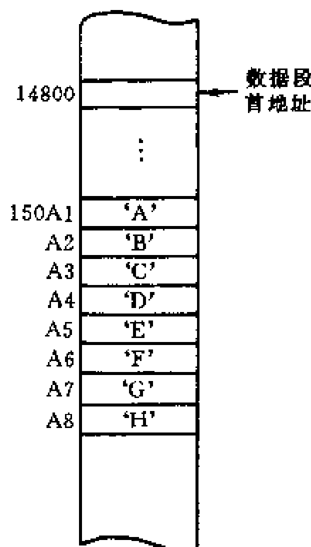


图 3-36

27. 选择题：

(1) 某微机具有 16M 字节的内存空间，其 CPU 的地址总线应有(　)条。

A. 26　　B. 28　　C. 20　　D. 22　　E. 24

(2) 8086/8088 CPU 要求加到 RESET 引脚上的复位正脉冲信号，其宽度至少要有(　)个时钟周期才能有效复位，如果是上电复位则要求正脉冲的宽度不少于(　)μs。

A. 4，50　　B. 5，60　　C. 4，70　　D. 5，80

(3) 当 RESET 信号进入高电平状态时，将使 8086/8088 CPU 的(　)寄存器初始化为 FFFFH。

A. SS　　B. DS　　C. ES　　D. CS

(4) 8086/8088 CPU 与慢速的存储器或 I/O 接口之间，为了使传送速度能匹配，有时需要在(　)状态之间插入若干等待周期 T_W。

A. T_1 和 T_2　　B. T_2 和 T_3　　C. T_3 和 T_4　　D. 随机

28. 填空题：

(1) 8086/8088 CPU 执行指令中所需操作数地址由(　)计算出(　)位偏移量部分送(　)，由(　)最后形成一个(　)位的内存单元物理地址。

(2) 8086/8088 CPU 在总线周期的 T_1，用来输出(　)位地址信息的最高(　)位，而在其它时钟周期，则用来输出(　)信息。

(3) 8086/8088 CPU 复位后，从(　)单元开始读取指令字节，一般这个单元在( )区中，在其中设置一条(　)指令，使 CPU 对系统进行初始化。

(4) 8086 系统的存储体系结构中，1M 字节存储体分(　)个库，每个库的容量都是(　)字节，其中和数据总线 D_{15}～D_8 相连的库全部由(　)单元组成，称为高位字节库，并用(　)作为此库的选通信号。

(5) 8086/8088 系统中，可以有(　)个段地址，任意相邻的两个段地址相距( ) 个存储单元。

(6) 用段基值及偏移地址来指明一内存单元地址称为(　)。

(7) 通常 8086/8088 CPU 中当 EU 执行一条占用很多时钟周期的指令时，或者在多处理器系统中在交换总线控制时会出现(　)状态。

(8) 在 8086/8088 最大方式系统中各微处理器都含有两条(　)引脚，其中(　)比(　)具有更高的优先级。

第四章　指令系统和寻址方式

指令系统是指微处理器能执行的各种指令的集合。微处理器的主要功能是由它的指令系统来体现的。不同的微处理器有不同的指令系统，其中每一条指令对应着处理器的一种基本操作，这在设计微处理器时就已决定了。

对于一台计算机，它只能识别由二进制编码表示的指令，称为机器指令。一条机器指令应包含两部分内容：一是要给出此指令要完成何种操作，这部分称为指令操作码部分；另一部分是要给出参与操作的对象是什么，此部分称为操作数部分。在指令中可以直接给出操作数的值或者操作数存放在何处，操作的结果应送往何处等信息。处理器可根据指令字中给出的地址信息求出存放操作数的地址——称为有效地址 EA(Effective Address)，然后对存放在有效地址中的操作数进行存取操作。指令中关于如何求出存放操作数有效地址的方法称为操作数的寻址方式。计算机按照指令给出的寻址方式求出操作数有效地址和存取操作数的过程，称为寻址操作。

指令中还有程序转移类指令。这类指令所指出的地址是程序转移到指定的转移地址，然后再依次顺序执行程序。这种提供转移地址的方法称为程序转移地址的寻址方式。

为了全面地理解和掌握 8086/8088 的指令系统，本章先介绍指令的寻址方式及机器语言的指令码格式，然后再分类介绍 8086/8088 的指令系统。

第一节　8086/8088 指令系统的寻址方式

一、操作数的种类

指令中操作的对象称为操作数。8086/8088 指令系统中操作数的种类分为两大类：

1. 数据操作数

这类操作数是与数据有关的操作数，即指令中操作的对象是数据。数据操作数又可分为：

(1)立即数操作数：指令中要操作的数据在指令中。

(2)寄存器操作数：指令中要操作的数据存放在指定的寄存器中。

(3)存储器操作数：指令中要操作的数据存放在指定的存储单元中。

(4)I/O 操作数：指令中要操作的数据来自或送到 I/O 端口。

2. 转移地址操作数

这类操作数是与程序转移地址有关的操作数。即指令中操作的对象不是数据，而是要转移的目标地址。它也可以分为立即数操作数、寄存器操作数和存储器操作数，即要转移的目标地址包含在指令中，或存放在寄存器中，或存放在存储单元之中。

对于数据操作数，有的指令有两个操作数，一个称为源操作数，在操作过程中其值不改变。另一个称为目标操作数，操作后一般被操作结果代替。有的指令只有一个操作数，或没有操作数。

对于转移地址操作数，其指令只有一个目标操作数，它是一个供程序转移的目标地址。

二、寻址方式

在地址指定方式下指令中提供操作数或操作数地址的方法称为寻址方式。换句话说，寻址方式是规定如何对指令中操作数字段进行解释以找到操作数的方法。根据操作数的种类，8086/8088 指令系统的寻址方式分为两大类：数据寻址方式和转移地址寻址方式。

(一)数据的寻址方式

数据寻址方式又可分为四种类型：

1. 立即数寻址方式

立即数寻址方式所提供的操作数直接包含在指令中，紧跟在操作码之后，它作为指令的一部分，这种操作数称为立即数。立即数可以是 8 位的，也可以是的 16 位的。如果是的 16 位数，则高位字节存放在高地址中，低位字节存放在低地址中。例如：

```
MOV  BL,80H
MOV  AX,1090H
```

则指令执行情况如图 4-1 所示。执行结果为：(BL)＝80H，(AX)＝1090H。

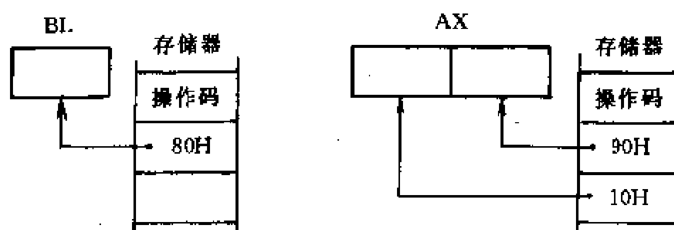


图 4-1 立即数寻址方式指令的执行情况

立即数寻址方式只能作为源操作数，主要用来给寄存器或存储单元赋值。

2. 寄存器寻址方式

寄存器寻址方式的操作数存放在指令规定的寄存器中，寄存器的名字可在指令中指出。对于 16 位操作数，寄存器可以是 AX、BX、CX、DX、SI、DI、SP 或 BP。对于 8 位操作数，寄存器可以是 AH、AL、BH、BL、CH、CL、DH 或 DL。例如：

```
MOV  CL,DL
MOV  AX,BX
```

如果(DL)＝50H，(BX)＝1234H，则执行结果为：(CL)＝50H，(AX)＝1234H。

寄存器寻址方式由于操作数就在寄存器中，不需要访问存储器来取得操作数，因而可以取得较高的运行速度，通常用于 CPU 内部操作。

3. 存储器寻址方式

存储器寻址方式的操作数存放在存储单元中，在指令中可以直接或间接给出存放操作数的地址，以达到存取操作数的目的。

(1)直接寻址方式

直接寻址方式的有效地址 EA(16 位偏移地址)在指令的操作码后面直接给出，它与指令的操作码一起，存放在存储器的代码段中，也是高位字节存放在高地址中，低位字节存放在低地址中。但是，操作数本身一般存放在存储器的数据段中。例如：

```
MOV  AL,[1064H]
```

如果(DS)＝2000H，则指令执行情况如图 4-2 所示。执行结果为：(AL)＝45H。

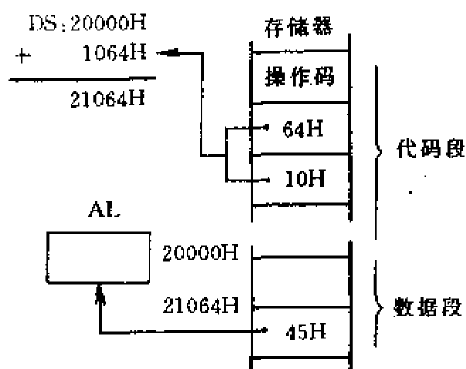


图 4-2 直接寻址方式指令的执行情况

注意这种直接寻址方式与前面已经介绍过的立即数寻址方式的不同。从指令的表示形式来看，在直接寻址方式中，对于表示有效地址的 16 位数，必须加上方括号。从指令的功能上来看，本例指令的功能不是将立即数 1064H 传送到累加器 AL，而是将一个有效地址是 1064H 的存储单元的内容传送到 AL。设此时数据段寄存器 DS＝2000H，则该存储单元的物理地址为：

$$
\begin{aligned}
2000H \times 10H + 1064H &= 20000H + 1064H \\
&= 21064H
\end{aligned}
$$

如果没有特殊指明，直接寻址方式的操作数一般在存储器的数据段，即隐含的段寄存器是 DS。但是 8086/8088 也允许段超越，即允许使用 CS、SS 或 ES 作为段寄存器，此时需要在指令中特别标明。方法是在有关操作数的前面写上段寄存器名，再加上冒号。例如，若以上指令改用 ES 作为段寄存器，则指令应表示成为以下形式：

```
MOV   AL,ES:[1064H]
```

在汇编语言指令中，可以用符号地址代替数值地址。例如：

```
MOV   AL,VALUE
```

或

```
MOV   AL,[VALUE]
```

此时 VALUE 为存放操作数单元的符号地址。

(2)寄存器间接寻址方式

寄存器间接寻址方式与前面已经讨论过的寄存器寻址方式不同，指令中指定的寄存器(是一个 16 位寄存器)的内容不是操作数，而是操作数的有效地址，操作数本身则在存储器中。

寄存器间接寻址方式可用的寄存器有四个：SI、DI、BX 和 BP。寄存器间接寻址方式的有效地址

$$
EA = \begin{bmatrix} (SI) \\ (DI) \\ (BX) \\ (BP) \end{bmatrix}
$$

但若选择其中不同的间址寄存器，默认的段寄存器有所不同。若选择 SI、DI、BX 寄存器，则存放操作数的段寄存器默认为 DS，若选择 BP 寄存器，则对应的段寄存器应为 SS。使用时应加以注意。

书写指令时，用作间址的寄存器必须加上方括弧，以免与一般的寄存器寻址方式混淆。例如：

```
MOV   AX,[SI]
MOV   [BX],AL
```

如果(DS)＝3000H，(SI)＝2000H，(BX)＝1000H，(AL)＝64H，指令执行情况如图 4-3 所示。执行结果为：(AX)＝4050H，(31000H)＝64H。

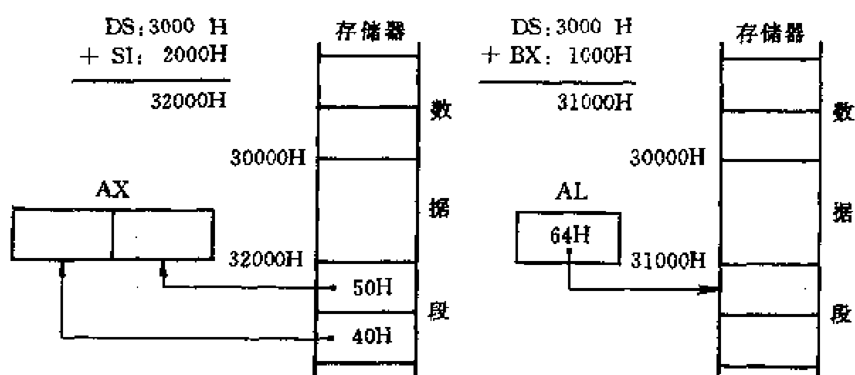


图 4-3 寄存器间接寻址方式指令的执行情况

无论用 SI、DI、BX 或者 BP 作为间址寄存器，都允许段超越。

例如：

```
MOV   ES:[DI],AX
MOV   DX,DS:[BP]
```

(3)寄存器相对寻址方式

寄存器相对寻址方式的有效地址 EA 是一个由指令中给定的 8 位或 16 位位移量 disp (displacement)和指定的寄存器内容相加之和。可用作寄存器相对寻址方式的寄存器有 SI、DI、BX 和 BP。即

$$EA=\begin{bmatrix}(SI)\\(DI)\\(BX)\\(BP)\end{bmatrix}+\begin{bmatrix}8\text{ 位 disp}\\16\text{ 位 disp}\end{bmatrix}$$

上述位移量可以看成是一个基于基值/变值的一个相对值，故称为寄存器相对寻址方式。位移量一般可用符号地址表示。

例如：

```
MOV   [SI+10H],AX
MOV   CX,[BX+COUNT]
```

如果(DS)=3000H，(SI)=2000H，(BX)=1000H，COUNT=1050H，(AX)=4050H，则指令执行情况如图 4-4 所示。执行结果为：(2010H)=4050H，(CX)=4030H。

寄存器相对寻址方式的操作数在汇编语言中书写时可以是下述形式之一：

```
MOV   AL,[BP+TABLE]
MOV   AL,[BP]+TABLE
MOV   AL,TABLE[BP]
```

其实以上三条指令实质上代表同一条指令。

在一般情况下，若指令中指定的寄存器是 SI、DI、BX，则存放操作数的段寄存器默认为 DS。若指令中指定 BP 寄存器，则对应的段寄存器应为 SS。

同样，寄存器相对寻址方式也允许段超越。

(4)基址变址寻址方式

基址变址寻址方式的有效地址是一个基址寄存器(BX 或 BP)和一个变址寄存器(SI 或 DI)的内容之和。即：

图 4-4 基址变址寻址方式指令的执行情况

$$EA=\begin{bmatrix}(SI)\\(DI)\end{bmatrix}+\begin{bmatrix}(BX)\\(BP)\end{bmatrix}$$

在汇编语言中书写时可以是下列形式之一：

MOV [BX+DI],AX

MOV AH,[BP][SI]

如果 (DS)=3000H,(SS)=4000H,(BX)=1000H,(DI)=1100H,(AX)=0050H,(BP)=2000H,(SI)=1200H,(43200H)=56H,则指令执行结果为:(32100H)=0050H,(AH)=56H。

在一般情况下,由基地址决定哪一个段寄存器作为地址指针。即若用 BX 作为基地址,则操作数在数据段中。若用 BP 作为基地址,则操作数在堆栈段中。但若在指令中规定了段超越,则可用其它寄存器作为地址基准。

(5)基址变址相对寻址方式

基址变址相对寻址方式的有效地址是由指令中指定的 8 位或 16 位位移量(disp)、一个基址寄存器内容和一个变址寄存器内容之和。即：

$$EA=\begin{bmatrix}(SI)\\(DI)\end{bmatrix}+\begin{bmatrix}(BX)\\(BP)\end{bmatrix}+\begin{bmatrix}8\text{ 位 disp}\\16\text{ 位 disp}\end{bmatrix}$$

同样,当基址寄存器为 BP 时,段寄存器应为 SS。也允许段超越。

例如：

MOV AH,[BX+DI+1234H]

MOV [BP+SI+DATA],CX

若 (DS)=4000H,(SS)=5000H,(BX)=1000H,(DI)=1500H,(BP)=2000H,(SI) = 1050H,(CX)=2050H,DATA=1000H,(4373451)=64H,则指令执行结果为:(AH)=64H,(54050H)=2050H。

基址加变址相对寻址方式也可以表示成几种不同的形式：

MOV AX,[BX+SI+COUNT]

MOV AX,COUNT[BX][SI]

MOV AX,[BX+COUNT][SI]

```
MOV   AX,[BX]COUNT[SI]
MOV   AX,[BX+SI]COUNT
MOV   AX,COUNT[SI][BX]
```

4. I/O 端口寻址方式

对于 I/O 端口寻址方式有以下两种：

(1)直接端口寻址方式

对这种寻址方式，端口地址用 8 位立即数(0～255)表示。例如：

```
IN   AL,21H
```

此指令表示从 I/O 端口地址为 21H 的端口中读取数据送到 AL 中。

(2)间接端口寻址方式

此时 I/O 端口的地址应事先存放在规定的 DX 寄存器中(0～65535)。

例如：

```
MOV   DX,255
OUT   DX,AL
```

前一条指令是将端口地址 255 送到 DX 寄存器，后一条指令表示将 AL 中的内容输出到地址由 DX 寄存器内容所指定的端口中。

(二)转移地址的寻址方式

在 8086/8088 指令系统中，有一组指令被用来控制程序的执行顺序。程序的执行是由 CS 和 IP 的内容所决定的。通常情况下，当 BIU 完成一次取指周期后，就自动改变 IP 的内容以指向下一条指令的地址。使程序按预先存放在程序存储器中的指令的次序，由低地址到高地址顺序执行。如需要改变程序的执行顺序，转移到所要求的指令地址再顺序执行时，可以安排一条程序转移指令，并按指令的要求修改 IP 内容或同时修改 IP 和 CS 的内容，从而将程序转移到指令所指定的转移地址。程序转移的寻址方式就是找出程序转移的地址。转移地址可以在段内，也可以跨段(称段间转移)。寻求转移地址的方法称为转移地址寻址方式，其有如下四种方式：

1. 段内直接寻址方式

段内直接寻址方式也称为相对寻址方式。转移的地址是当前的 IP 内容和指令规定的 8 位或 16 位位移量之和。当位移量是 8 位时，称为短程转移；位移量是 16 位时称为近程转移。这种寻址方式适用于无条件转移或条件转移类指令。但条件转移只有 8 位位移量的短程转移。指令的格式表示为：

```
JMP   NEAR PTR PROGIA
JMP   SHORT QUEST
```

其中，PROGIA 和 QUEST 均为转向的符号地址，在机器指令中，用位移量来表示。在汇编语言中，如果位移量为 16 位，则在符号地址前加操作符 NEAR PTR，如果位移量为 8 位，则在符号地址前加操作符 SHORT。

2. 段内间接寻址方式

程序转移的地址存放在寄存器或存储单元中。这个寄存器或存储单元的内容可以用数据寻址方式中除了立即数以外的任何一种寻址方式取得，所得到的转移的有效地址用来更新 IP 的内容。由于此寻址方式仅修改 IP 的内容，所以这种寻址方式只能在段内进行程序转移。

这种寻址方式以及以下的两种段间寻址方式都不能用于条件转移指令。也就是说，条件转

移指令只能使用段内直接寻址的 8 位位移量。

段内间接寻址转移指令的格式可以表示为：

JMP　BX

JMP　WORD PTR [BP+TABLE]

等。其中 WORD PTR 为操作符，用以指出其后的寻址方式所取得的转向地址是一个字的有效地址，也就是说它是一种段内转移。

3. 段间直接寻址方式

这种寻址方式是在指令中直接给出 16 位的段基值和 16 位的偏移地址用来更新当前的 CS 和 IP 的内容。指令的格式可以表示为：

JMP　FAR PTR NEXTROUTINT

其中，NEXTROUTINT 为转向的符号地下，FAR PTR 则是表示段间转移的操作符。

4. 段间间接寻址方式

这种寻址方式是由指令中给出的存储器数据寻址方式字节求出存放转移地址的连续两个字的地址。其低位字地址单元存放的是偏移地址，高位字地址单元中存放的是转移段基值。这样即更新了 IP 内容又更新了 CS 的内容，故称为段间间接寻址。这种指令的格式可以表示为：

JMP　DWORD PTR[INTERS+BX]

其中，[INTERS+BX]说明数据寻址方式为寄存器相对寻址方式，DWORD PTR 为双字操作符，说明此指令是转向地址需取双字的段间转移指令。

第二节　8086/8088 指令码格式

我们用汇编语言编写的汇编语言程序输入计算机后，由机器提供的"汇编程序"将它翻译成由机器指令（指令码）组成的机器语言程序，才能由计算机识别并执行。因此汇编语言程序需由汇编程序翻译成可执行的机器语文程序，一般说来，这一过程不必由人来干预。我们这里只介绍一下基本原理，以便在必要时也可以用手工完成类似的工作。

8086/8088 指令系统的指令类型较多，功能很强。各种指令由于功能不同，需要指令码提供的信息也不同。为了满足不同功能的要求又要减少指令所占的空间，8086/8088 指令系统采用了一种灵活的、由 1～6 个字节组成的变字长的指令格式，包括操作码、寻址方式以及操作数三个部分。如图 4-5 所示。

通常指令的第一字节为操作码，规定指令的操作类型。第二字节规定操作数的寻址方式。接着以后的 3～6 字节依据指令的不同而取舍。可变字长的指令主要体现在这里，一般由它指出存储器操作数地址的位移量或立即数。

操作码/寻址方式字节格式如下：

操	作	码				D	W	MOD		REG			R/M		
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

第一字节中，W——指示操作数类型：W=0 为字节，W=1 为字；D——指示操作数的传送方向：D=0 寄存器操作数为源操作数，D=1 寄存器操作数为目标操作数。

第二字节指出所用的两个操作数存放的位置，以及存储器中操作数有效地址 EA 的计算方法。其中：

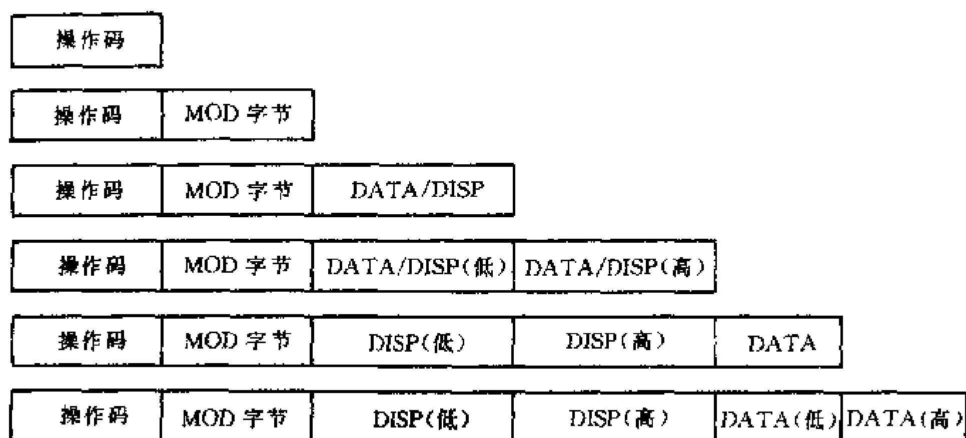


图 4-5　8086/8088 不同字长的指令码格式

REG 字段规定一个寄存器操作数，它作为源操作数还是目标操作数已由第一个字节中的 D 位规定。由 REG 字段选择寄存器的具体规定见表 4-1 所示。

MOD 字段用来区分另一个操作数在寄存器中(寄存器寻址)还是在存储器中(存储器寻址)。在存储器寻址的情况下,还用来指出该字节后面有多少偏移量字节(即指出存储器操作数地址偏移量的字节数)。MOD 字段编码表如表 4-2 所示。

R/M 字段受 MOD 字段控制。若 MOD=11 为寄存器方式,R/M 字段将指出第二操作数所在寄存器编号;MOD=00,01,10 为存储器方式,R/M 则指出如何计算存储器中操作数地址。MOD 与 R/M 字段组合的寻址方式见表 4-3。

表 4-1　REG 字段编码表

REG	W=1(字操作)	W=0(字节操作)
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

表 4-2　MOD 字段编码表

MOD	寻址方式
00	存储器寻址,没有位移量
01	存储器寻址,有 8 位位移量
10	存储器寻址,有 16 位位移量
11	寄存器寻址,没有位移量

表 4-3　各种 MOD 与 R/M 字段组合编码及有关地址的计算

MOD=11 寄存器寻址			MOD≠11 存储器寻址,有效地址的计算公式			
R/M	W=1	W=0	MOD R/M	不带位移量 00	带 8 位位移量 01	带 16 位位移量 10
000	AX	AL	000	[BX+SI]	[BX+SI+D8]	[BX+SI+D16]
001	CX	CL	001	[BX+DI]	[BX+DI+D8]	[BX+DI+D16]

（续）

MOD=11 寄存器寻址			MOD≠11 存储器寻址，有效地址的计算公式			
R/M	W=1	W=0	MOD R/M	不带位移量 00	带 8 位位移量 01	带 16 位位移量 10
010	DX	DL	010	[BP+SI]	[BP+SI+D8]	[BP+SI+D16]
011	BX	BL	011	[BP+DI]	[BP+DI+D8]	[BP+DI+D16]
100	SP	AH	100	[SI]	[SI+D8]	[SI+D16]
101	BP	CH	101	[DI]	[DI+D8]	[DI+D16]
110	SI	DH	110	（直接寻址）	[BP+D8]	[BP+D16]
111	DI	BH	111	[BX]	[BX+D8]	[BX+D16]

为了清楚地了解指令格式，现举例如下：

例 1：

MOV　AH,[BX+DI+50H]

代码格式：

OPCODE	D	W	MOD	REG	R/M	disp—8
100010	1	0	01	100	001	01010000

指令码为：8A6150H。

例 2：

ADD　DISP[BX][DI],DX ;DISP=4523H

代码格式：

OPCODE	D	W	MOD	REG	R/M	disp—L0	disp—Hi
000000	0	1	10	010	001	01000101	00100011

指令码为：01914523H。

第三节　8086/8088 指令系统

8086/8088 的指令系统大致可分成以下六种类型：

① 数据传送指令；

② 算术运算指令；

③ 位操作指令；

④ 串操作指令；

⑤ 程序控制指令；

⑥ 处理器控制指令。

学习指令系统着重要掌握指令的基本操作功能、合法的寻址方式以及对状态标志位的影响。

一、数据传送指令

数据传送指令是将数据、地址或立即数传送到寄存器或存储单元中。它又可分为通用数据传送指令、输入输出指令、目的地址传送指令和标志传送指令等四组。

数据传送指令对状态标志位不发生影响，只有第四组中的两条涉及标志寄存器 FLAG 的指令(SAHF 和 POPF)例外。下面分别进行讨论。

(一)通用数据传送指令

1. 数据传送指令

指令格式及操作：

```
MOV dst,src              ;(dst)←(src)
```

指令中的 dst 表示目标操作数，src 表示源操作数，指令实现的操作是将源操作数传送到目标操作数地址。这种传送实际上是进行数据的"复制"，将源操作数复制到目标操作数地址中去，源操作数本身不变。

这种双操作数指令在汇编语言中的表示方法，总是将目标操作数写在前面，源操作数写在后面，二者之间用一个逗号隔开。

在 MOV 指令中源操作数可以为：存储器、寄存器、段寄存器和立即数；目标操作数可以为：存储器、寄存器(不能为 IP)和段寄存器(不能为 CS)。除了目标操作数和源操作数不能同时为存储器、段寄存器、立即数和段寄存器以外，可以任意搭配。数据传送的方向如图 4-6 所示。

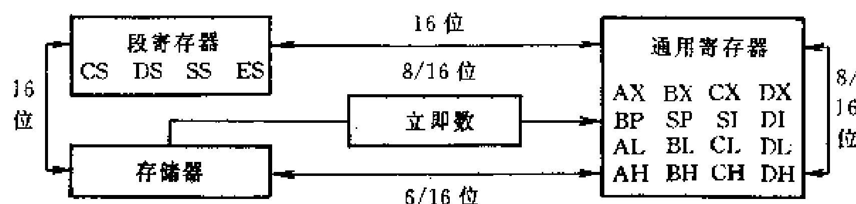


图 4-6 MOV 指令数据传送方向

需说明一点，对于代码段寄存器 CS 和指令指针寄存器 IP，通常无需用户利用传送指令改变其中的内容。但是 CS 可以作为源操作数。

2. 堆栈操作指令

堆栈操作指令是用来完成压入和弹出堆栈操作的。8086/8088 指令系统中提供了完成这两种操作的相应指令。

(1)压入堆栈指令

指令格式及操作：

```
PUSH  src ; (SP)←(SP)-2,((SP)+1:(SP))←(src)
```

指令完成的操作是"先移后入"，即先将堆栈指针 SP 减 2，然后再将操作数 src 压入由 SP 指出的栈顶中。指令中的操作数 src 可以是通用寄存器和段寄存器，也可以是由某种寻址方式所指示的存储单元，但不能是立即数。例如：

```
PUSH  AX ; (SP)←(SP)-2,((SP)+1)←(AH),((SP))←(AL)
PUSH  CS
PUSH  [SI]
```

(2)弹出堆栈指令

指令格式及操作：

POP　dst ；(dst)←((SP)+1:(SP))，(SP)←(SP)+2

指令完成的操作是"先出后移"，即先将堆栈指针 SP 所指示的栈顶存储单元的值弹出到操作数 dst 中，然后再将堆栈指针 SP 加 2。指令中的操作数 dst 可以是存储器、通用寄存器或段寄存器(但不能是代码段寄存器 CS)，同样，不能是立即数。例如：

```
POP  BX ；(BH)←((SP)+1)，(BL)←((SP))，(SP)←(SP)+2
POP  ES
POP  MEM[DI]
```

应该注意，堆栈操作指令中的操作数类型必须是字操作数(16 位)。

3. 数据交换指令

指令格式及操作：

XCHG　opr1，opr2 ；(opr1)←→(opr2)

这是一条交换指令，它的操作是使源操作数与目标操作数进行交换，即不仅将源操作数传送到目标操作数，而且，同时将目标操作数传送到源操作数。

交换指令的源操作数和目标操作数各自均可以是寄存器或存储器，但不能二者同时为存储器。也就是说可以在寄存器与寄存器之间，或者寄存器与存储器之间进行交换，但是不能在存储器的存储单元之间交换。此外，段寄存器的内容不能参加交换。

交换的内容可以是一个字节(8 位)，也可以是一个字(16 位)。

4. 字节转换指令

指令格式及操作：

XLAT　src-table ；(AL)←((BX)+(AL))

XLAT　指令是字节查表转换指令，可以根据表中元素的序号，查出表中相应元素的内容。为了实现查表转换，预先应将表的首地址，即表头地址传送到 BX 寄存器，元素的序号即位移量送 AL，表中第一个元素的序号为 0，然后依次是 1，2，3，…。执行 XLAT 指令后，表中指令序号的元素存于 AL。由于借助了 AL 寄存器进行，所以被寻址的表的最大长度为 255 个字节。这是一种特殊的基址变址寻址方式，基址寄存器为 BX，变址寄存器为 AL。利用 XLAT 指令实现不同数制或编码系统之间的转换十分方便。

例如内存的数据段有一张 16 进制数的 ASCII 码表，其首地址为 Hex-table，如图 4-7 所示，为欲查出第 10 个元素(元素序号从 0 开始)，即"A"的 ASCII 码，则可用以下几条指令实现：

```
MOV   BX,OFFSET Hex—table     ;(BX)←表首址
MOV   AL,0AH                  ;(AL)←序号
XLAT  Hex—table               ;查表转换
```

结果"A"的 ASCII 码在 AL 中，即(AL)=41H。

上例中查表转换指令后面的操作数首地址 Hex—table(类型为字节)，实际上已经预先传送到了 BX 寄存器中，写在 XLAT 指令中是为了用汇编程序检查类型的正确性。但是 XLAT 指令后面也可以不写操作数。

BX 寄存器中包含着表的首地址，所在的段由隐含值确定。但也允许段超越，此时必须在指令中写明重设的段寄存器。XLAT 指令的几种表示形式如下：

	存储器
	⋮
Hex—table	30H('0')
Hex—table+1	31H('1')
Hex—table+2	32H('3')
	⋮
Hex—table+9	39H('9')
Hex—table+10	41H('A')
Hex—table+11	42H('B')
	⋮
Hex—table+15	46H('F')
	⋮

图 4-7　16 进制数的 ASCII 码表

```
XLAT                ;不写操作数
XLAT   src-table    ;写操作数
XLATB               ;B 表示字节类型,不允许再写操作数
XLAT   ES:src-table ;重设段寄存器为 ES
```

（二）输入输出指令

输入输出指令共有两条。输入指令 IN 用于从外设端口接收数据，输出指令 OUT 则向端口发送数据。无论是接收到的数据或是准备发送的数据都必须在累加器 AL(字节)或 AX(字)中，所以这是两条累加器专用指令。

输入输出指令可以分为两大类：一类是端口直接寻址的输入输出指令；另一类是端口通过 DX 寄存器间接寻址的输入输出指令。在直接寻址的指令中只能寻址 255 个端口(0～255)，而间接寻址的指令中可寻址 64K 个端口(0～65535)。

1. 输入指令

(1)直接寻址的输入指令

指令格式及操作：

```
IN   acc,port ; (acc)←(port)
```

此指令是将 8/16 位数据直接经输入端口 port(地址 0～255)送入 AL/AX 累加器中。

(2)间接寻址的输入指令

指令格式及操作：

```
IN   acc,DX         ; (acc)←((DX))
```

此指令是从 DX 寄存器内容指定的端口中将 8/16 位数据送入 AL/AX 累加器中。这种寻址方式的端口地址由 16 位地址表示，执行此指令前应将 16 位地址存入 DX 寄存器中。

2. 输出指令

(1)直接寻址的输出指令

指令格式及操作：

```
OUT   port,acc      ; (port)←(acc)
```

此指令是从 AL(8 位)或 AX(16 位)累加器输出 8/16 位数据到指令指定的 I/O 端口中。

(2) 间接寻址的输出指令

指令格式及操作：

```
OUT   DX,acc        ; ((DX))←(acc)
```

此指令是从 AL(8 位)或 AX(16 位)累加器中输出 8/16 位数据到由 DX 寄存器内容指定的 I/O 端口中。

（三）目的地址传送指令

8086/8088CPU 提供了三条把地址指针写入寄存器或寄存器对的指令，它们可以用来写入近地址指针和远地址指针。

1. 取有效地址指令

指令格式：

```
LEA    reg16,mem
```

LEA 指令将一个近地址指针写入到指定的寄存器。指令中的目标操作数必须是一个 16 位通用寄存器，源操作数必须是一个存储器操作数，指令的执行结果是把源操作数的有效地址，即 16 位位移地址传送到目标寄存器。例如：

```
LEA    BX,BUFFER
LEA    AX,[BP][DI]
LEA    DX,BETA[BX][SI]
```

注意 LEA 指令与 MOV 指令的区别，比较下面两条指令：

```
LEA    BX,BUFFER
MOV    BX,BUFFER
```

前者将存储器 BUFFER 的位移地址送到 BX，而后者将存储器 BUFFER 的内容(两个字节)传送到 BX。当然也可以用 MOV 指令来得到存储器的位移地址，例如以下两条指令的效果相同：

```
LEA    BX,BUFFER
MOV    BX,OFFSET BUFFER
```

其中 OFFSET BUFFER 表示存储器 BUFFER 位移地址。

(2)地址指针装入 DS 指令

指令格式：

```
LDS    reg16,mem32
```

LDS 指令和下面即将介绍的 LES 指令都是用于写入远地址指针。源操作数可以是任一个存储器，目标操作数可以是任一个 16 位通用寄存器。

LDS 传送一个 32 位的远地址指针，其中包括一个位移地址和一个段基值，前者送指定寄存器，后者送数据段寄存器 DS。例如：

```
LDS    SI,[0010H]
```

设原来 DS＝C000H，而有关存储单元的内容为(C0010H)＝80H，(C0011H)＝01H，(C0012H)＝00H，(C0013H)＝20H，则执行以上指令后，SI 寄存器的内容为 0180H，段寄存器 DS 的内容为 2000H。

3. 地址指针装入 ES 指令

指令格式：

```
LES    reg16,mem32
```

LES 指令与 LDS 类似，也是装入一个 32 位的远地址指针。位移地址送指定寄存器，但是，段基值送附加段寄存器 ES。

目的地址传送指令常常用于在串操作时建立初始的地址指针。

(四)标志传送指令

CPU 中有一标志寄存器 FLAG，其中每一状态标志位表示 CPU 运行的状态。许多指令执行结果会影响标志寄存器的某些状态标志位。同时，有些指令的执行也受标志寄存器中某些位的控制。标志传送指令共有四条。这些指令都是单字节指令，指令的操作数以隐含形式规定(隐含的操作数是 AH 寄存器)。

1. 取标志指令

指令格式：

LAHF

LAHF 指令将标志寄存器 FLAG 中的五个状态标志位，SF、ZF、AF、PF 以及 CF 分别取出传送到累加器 AH 的对应位，如图 4-8 所示。

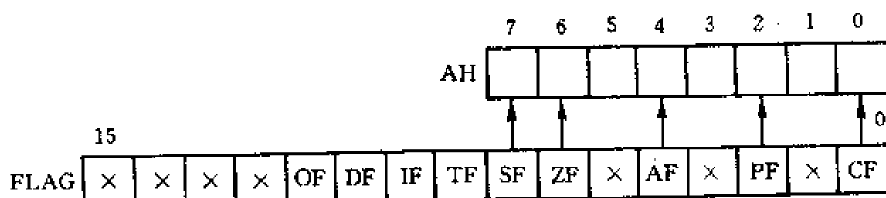


图 4-8 LAHF 指令操作示意图

LAHF 指令对状态标志位没有影响。

2. 置标志指令

指令格式：

SAHF

SAHF 指令的传送方向与 LAHF 相反，将 AH 寄存器中的第 7、6、4、2、0 位分别传送到标志寄存器的对应位。

SAHF 指令将影响状态标志位，FLAG 寄存器中的 SF、ZF、AF、PF 和 CF 将被修改成 AH 寄存器对应位的状态，但其余状态标志位即 OF、DF、IF 和 TF 不受影响。

3. 标志压入堆栈指令

指令格式及操作：

PUSHF ;(SP)←(SP)－2,((SP)＋1;(SP))←(FLAG)

PUSHF 指令先将 SP 减 2，然后将标志寄存器 FLAG 的内容(16 位)压入堆栈。这条指令本身不影响状态标志位。

4. 标志弹出堆栈指令

指令格式及操作：

POPF ;(FLAG)←((SP)＋1;(SP)),(SP)←(SP)＋2

POPF 指令的操作与 PUSHF 相反，它将堆栈内容弹出到标志寄存器，然后 SP 加 2。POPF 指令对状态标志位有影响，使各状态标志位恢复成为压入堆栈以前的状态。

PUSHF 和 POPF 指令可用于保护调用过程以前标志寄存器的值。过程返回以后再恢复这些标志状态。

二、算术运算指令

(一)算术运算的数据类型

8086/8088 的算术运算指令可以处理四种类型的数：无符号的二进制数、带符号的二进制数、无符号压缩十进制(压缩型 BCD 码)数和无符号的非压缩十进制(非压缩型 BCD 码)数。除压缩十进制数只有加/减运算外，其余三种数据类型都可以进行加、减、乘、除运算。

二进制的无符号数和带符号数的长度都可以是 8 位或 16 位，但应注意它们所能表示的数的范围是不同的。若是带符号数，则用补码表示。

十进制数以字节的形式存储。对压缩十进制数，每个字节存两位数，即两位 BCD 码。因而对于一个字节来说，压缩十进制数的范围是 0～99。而对非压缩的十进制数，每个字节存一位数，即由字节的低 4 位决定存放的数字。高 4 位，在进行乘/除运算时必须全为 0。加/减运算时

可以是任何值。

8086/8088 提供了各种调整操作指令。故除了可以对二进制数进行算术运算以外，也可以方便地进行压缩的或非压缩的十进制数的算术运算。

（二）算术运算指令对标志的影响

8086/8088 的算术运算指令将运算结果的某些特性传送到 6 个标志上去，这些标志中的绝大多数可由跟在算术运算指令后的条件转移指令进行测试，以改变程序的流程。因而掌握指令结果对标志的影响，对编程有着重要的作用。关于 6 个标志的含义已在第三章阐述了，这里不在重复。

算术运算类指令共有 20 条，包括加、减、乘、除运算、符号扩展和十进制调整指令，除符号扩展指令(CBW,CWD)外，其余指令都影响标志。

（三）二进制数运算指令

1. 加法指令

加法指令包括不带进位加法指令、带进位加法指令和加 1 指令。

(1)加法指令

指令格式及操作：

```
ADD  dst,src          ; (dst)←(dst)+(src)
```

ADD 指令将目标操作数与源操作数相加，并将结果存回目标操作数。加法指令将影响状态标志位。

目标操作数可以是寄存器或存储器，源操作数可以是寄存器、存储器或立即数。但是源操作数和目标操作数不能同时为存储器。另外，不能对段寄存器进行加法运算(段寄存器也不能参加减法、乘法和除法运算)。加法指令的操作对象可以是 8 位数(字节)，也可以是 16 位数(字)。例如：

```
ADD  CL,10
ADD  DX,SI
ADD  AX,MEM
ADD  DATA[BX],AL
ADD  ALPHA[DI],30H
```

相加的数据类型可以根据编程者的意图，规定为带符号数或无符号数。对于带符号数，如果相加结果超出范围，则发生溢出。例如：

```
MOV  AL,7EH
MOV  BL,5BH
ADD  AL,BL
```

执行以上三条指令以后，相加结果(AL)=D9H，此时各状态标志位的状态为：SF=1，ZF=0，AF=1，PF=0，CF=0，OF=1。其中 OF=1 表示发生了溢出，这是由于相加结果超过了 127。但最高位并未产生进位，故 CF=0。

(2)带进位加法指令

指令格式及操作：

```
ADC  dst,src          ; (dst)←(dst)+(src)+(CF)
```

ADC 指令是将目标操作数与源操作数相加，再加上进位标志 CF 的内容，然后将结果返回目标操作数。与 ADD 指令一样，ADC 指令的运算结果也将修改状态标志位。目标操作数及

源操作数的类型与 ADD 指令相同，而且 ADC 指令同样也可以进行字节操作或字操作。

带进位加法指令主要用于多字节数据的加法运算。如果低字节相加时产生进位，则在下一次高字节相加时将这个进位加进去。

例 4.1　要求计算两个多字节 16 进制数之和：3B74AC60F8H＋20D59E36C1H＝？

式中被加数和加数均有五个字节，可以编一个循环程序实现以上运算。假设已将被加数和加数分别存入从 DATA1 和 DATA2 开始的两个内存区，且均为低位字节在前，高位字节在后，如图 4-9 所示。要求相加所得结果仍存回以 DATA1 为首址的内存区。

运算程序流程图见图4-10。程序见下：

```
          MOV  CX,5            ;设置循环次数
          MOV  SI,0            ;置位移量初值
          CLC                  ;清进位
LOOPER:   MOV  AL,DATA2[SI]    ;取一个加数
          ADC  DATA1[SI],AL    ;和另一个加数相加
          INC  SI              ;位移量加 1
          DEC  CX              ;循环次数减 1
          JNZ  LOOPER          ;加完否，若没完，转 LOOPER，继续相加
          HLT                  ;程序暂停
```

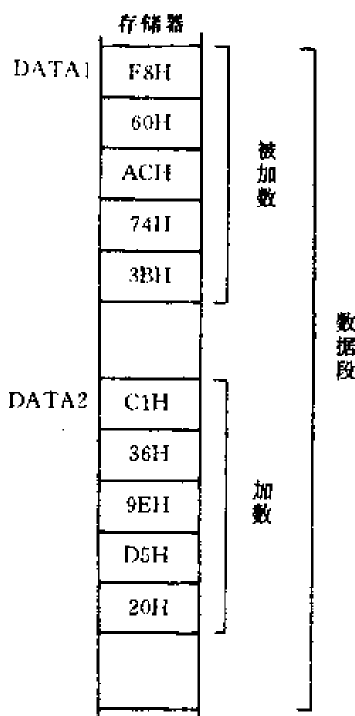


图 4-9　例 4.1 中被加数和加数在内存中的存放情况

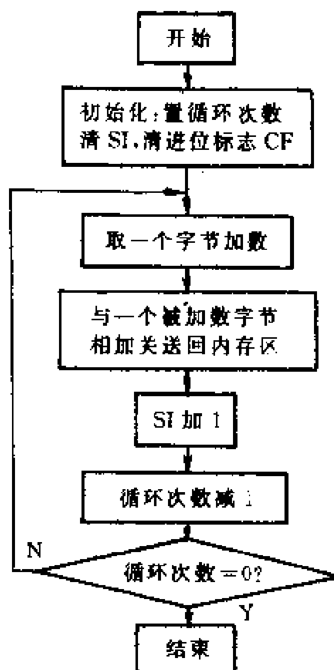


图 4-10 例 4.1 的流程图

由于程序中使用了带进位加指令，因此在循环程序开始之前应将进位标志 CF 清零（用 CLC 指令），以免在第一次相加最低位字节时因原来的进位标志状态影响相加结果而产生错误。

（3）加 1 指令

指令格式及操作：

INC　dst ；(dst)←(dst)+1

INC 指令将目标操作数加 1。指令将影响状态标志位，如 SF、ZF、AF、PF 和 OF 但对进位标志 CF 没有影响。

INC 指令中目标操作数可以是寄存器或存储器，但不能是段寄存器。其类型为字节操作或字操作均可。例如：

```
INC   DL
INC   SI
INC   BYTE PTR[BX][SI]
INC   WORD PTR[DI]
```

指令中的 BYTE PTR 或 WORD PTR 分别指定随后的存储器操作数的类型是字节或字。INC 指令常常用于循环程序中修改地址。

2. 减法指令

减法指令包括不带借位减法指令、带借位减法指令、减 1 指令、求补指令和比较指令。

(1) 减法指令

指令格式及操作：

SUB　dst, src　　　；(dst)←(dst)−(src)

SUB 指令用目标操作数减源操作数，结果送回目标操作数。指令对状态标志位有影响。

操作数的类型与加法指令一样，即目标操作数可以是寄存器或存储器，源操作数可以是立即数、寄存器或存储器，但不允许两个存储器相减，既可以字节相减，也可以字相减。例如：

```
SUB   AL,37H
SUB   DX,BX
SUB   CX,VARE1
SUB   ARRAY[DI],AX
SUB   BETA[BX][DI],512
```

减法数据的类型也可以根据程序员的要求约定为带符号数或无符号数。当无符号数的较小数减较大数时，因不够减而产生借位，此时进位标志 CF 置 1。当带符号数的较小数减较大数时，将得到负的结果，则符号标志 SF 置 1。带符号数相减如果结果溢出，则 OF 置 1。

(2)带借位减法指令

指令格式及操作：

SBB　dst,src　　　；(dst)←(src)−(CF)

SBB 指令是将目标操作数减源操作数，然后再减进位标志 CF，并将结果送回目标操作数。SBB 指令对标志的影响与 SUB 指令相同。

目标操作数及源操作数的类型也与 SUB 指令相同。8 位或 16 位数运算均可。例如：

```
SBB   BX,1000
SBB   CX,DX
SBB   AL,DATA1[SI]
SBB   DISP[BP],BL
SBB   [SI+6],97
```

带借位减指令主要用于多字节的减法。

(3)减 1 指令

指令格式及操作：

```
DEC  dst ; (dst)←(dst)－1
```

DEC 指令将目标操作数减 1。指令对状态标志位 SF、ZF、AF、PF 和 OF 有影响，但不影响进位标志 CF。

操作数与 INC 一样，可以是寄存器或存储器(段寄存器不可)。其类型是字节操作或字操作均可。例如：

```
DEC  BL
DEC  CX
DEC  BYTE PTR[BX]
DEC  WORD PTR[BP][DI]
```

在循环程序中常常利用 DEC 指令来修改循环次数。例如：

```
        MOV  AX,0FFFFH
CYC:    DEC  AX
        JNZ  CYC
        HLT
```

以上程序段中 DEC　AX 指令重复执行 65535(0FFFFH)次。此程序实际上是一段延时程序。

(4)求补指令

指令格式及操作：

```
NEG  dst            ; (dst)←0－(dst)
```

NEG 指令的操作是用"0"减去目标操作数，结果送回原来的目标操作数。求补指令对状态标志位有影响。

操作数可以是寄存器或存储器。可以对 8 位数或 16 位数求补。例如：

```
NEG  BL
NEG  AX
NEG  BYTE PTR[BP][SI]
NEG  WORD PTR[DI+20]
```

利用 NEG 指令可以得到负数的绝对值。例如假设原来 AL＝0FFH(0FFH 是－1 的补码)，执行指令 NEG AL 后，结果为 AL＝01。

例 4.2　内存数据段存放了 100 个带符号数，首地址为 AREA1，要求将各数取绝对值后存入以 AREA2 为首址的内存区。

由于 100 个带符号数中可能既有正数；又有负数，因此先要判断正负。如为正数，可以原封不动地传送到另一内存区；如为负数，则需先求补即可得到负数的绝对值，然后再传送。程序如下：

```
        LEA  SI,AREA1         ; (SI)←源地址指针
        LEA  DI,AREA2         ; (DI)←目的地址指针
        MOV  CX,100           ; (CX)←循环次数
CHECK:  MOV  AL,[SI]          ; 取一个带符号数到 AL
        OR   AL,AL            ; AL 内容不变，但使之影响标志
```

```
        JNS   NEXT              ；若(SF)=0,则转 NEXT
        NEG   AL                ；否则求补
NEXT:   MOV [DI],AL             ；传送到目的地址
        INC   SI                ；源地址加 1
        INC   DI                ；目的地址加 1
        DEC   CX                ；循环次数减 1
        JNZ   CHECK             ；如不等于零,则转 CHECK
        HLT                     ；停止
```

(5)比较指令

指令格式及操作：

```
CMP   dst,src      ；(dst)-(src)
```

CMP 指令用目标操作数减源操作数，但结果不送回目标操作数。因此，执行比较指令以后，被比较的两个操作数内容均保持不变，而比较结果反映在状态标志位上，这是比较指令与减法指令 SUB 的区别所在。

CMP 指令的目标操作数可以是寄存器或存储器，源操作数可以是立即数、寄存器或存储器，但不能同时为存储器。可以进行字节比较，也可以进行字比较。例如：

```
CMP   AL,0AH        ；寄存器与立即数比较
CMP   CX,DI         ；寄存器与寄存器比较
CMP   AX,AREA1      ；寄存器与存储器比较
CMP   [BX+5],SI     ；存储器与寄存器比较
CMP   GAMMA,100     ；存储器与立即数比较
```

比较指令的执行结果将影响状态标志位。例如，若两个被比较的内容相等，则(ZF)=1。又如，假设被比较的两个无符号数中，前者小于后者(即不够减)，则(CF)=1 等等。比较指令常常与条件转移指令结合起来使用，完成各种条件判断和相应的程序转移。

例 4.3　在数据段从 DATA 开始的存储单元中分别存放了两个 8 位无符号数，试比较它们的大小，并将大者传送到 MAX 单元。可编程如下：

```
        LEA   BX,DATA       ；DATA 偏移地址送 BX
        MOV   AL,[BX]       ；第一个无符号数送 AL
        INC   BX            ；BX 指向第二个无符号数
        CMP   AL,[BX]       ；两个数比较
        JNC   DONE          ；如(CF)=0,则转 DONE
        MOV   AL,[BX]       ；否则,第二个无符号数送 AL
DONE:   MOV   MAX,AL        ；较大的无符号送 MAX 单元
        HLT                 ；停止
```

3. 乘法指令

8086/8088 指令系统中有两条乘法指令，可以实现无符号数的乘法和带符号数的乘法，它们都只有一个源操作数，而目标操作数是隐含的。这两条指令都可以实现字节或字的乘法运算。进行乘法运算时，如果两个 8 位数据相乘，那么其乘积最多为 16 位；如果两个 16 位数据相乘，可得到 32 位的乘积。

需要指出的是 8086/8088 CPU 在执行乘法指令时，有一个操作数总是放在累加器(8 位数放在 AL，16 位数放在 AX)中。8 位数相乘，其乘积 16 位存放在 AX 中。16 位数相乘，其乘积为 32 位被存放在 DX 和 AX 中，其中高 16 位存于 DX 中，低 16 位存于 AX 中。如图 4-11 所示。

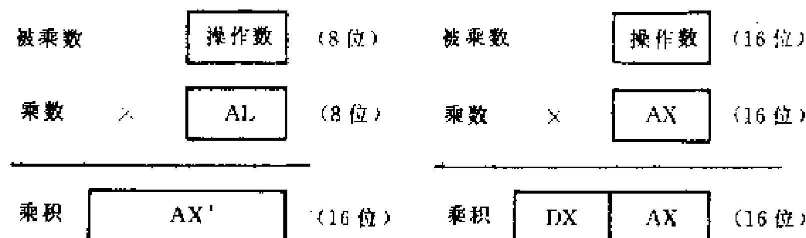


图 4-11　乘法运算的操作数及运算结果

(1)无符号数乘法指令

指令格式及操作：

```
MUL  src   ;(AX)←(src)×(al)        (字节乘法)
           ;(DX:AX)←(src)×(ax)     (字乘法)
```

MUL 指令对状态标志位 CF 和 OF 有影响，SF、ZF、AF 和 PF 不确定。

MUL 指令执行 8 位或 16 位无符号数的乘法。一个操作数(乘数)在累加器中(8 位乘法时乘数在 AL，16 位乘法时乘数在 AX)是隐含的。另一个操作数 src (被乘数)必须在寄存器或存储单元中。两个操作数均按无符号数处理，它们的取值范围为 0～255(字节)、或 0～65535(字)。例如：

```
MUL  AL                    ;AL 乘 AL
MUL  BX                    ;AX 乘 BX
MUL  BYTE PTR[DI+6]        ;AL 乘存储器(8 位)
MUL  WORD PTR ALPHA        ;AX 乘存储器(16 位)
```

两个 8 位数相乘，乘积可能有 16 位，结果存放在 AX 中；两个 16 位数相乘，乘积可能有 32 位，存放在 DX(高 16 位)和 AX(低 16 位)中。如果运算结果的高半部分(在 AH 或 DX 中)为零，则状态标志位(CF)=(OF)=0，否则(CF)=(OF)=1。因此，状态标志位(CF)=(OF)=1，表示 AH 或 DX 中包含着乘积的有效数字。例如：

```
MOV  AL,14H          ;(AL)=14H
MOV  CL,05H          ;(CL)=05H
MUL  CL              ;(AX)=0064H,(CF)=(OF)=0
```

本例中结果的高半部分(AH)=0，因此，状态标志位(CF)=(OF)=0。

有了乘法(和除法)指令，使有些运算程序的编程变得简单方便。但是必须注意，乘法指令的执行速度很慢，除法指令也是如此。

(2)带符号数的乘法

指令格式：

```
IMUL  src        ;(AX)←(src)×(AL)        (字节乘法)
                 ;(DX:AX)←(src)×(AX)     (字乘法)
```

IMUL 指令对状态标志位的影响以及操作过程与 MUL 指令相同。但 IMUL 指令进行带符号数乘法时，指令将两个操作数均按带符号数处理。这是它与 MUL 指令的区别所在。8 位

和16位带符号数的取值范围分别是－128～＋127(字节)和－32768～＋32767(字)。如果乘积的高半部分仅仅是低半部分符号位的扩展,则状态标志位(CF)＝(OF)＝0;否则,高半部分包含乘积的有效数字而不只是符号的扩展部分,则(CF)＝(OF)＝1。例如：

```
MOV   AX,04E8H          ;(AX)=04E8H
MOV   BX,4E20H          ;(BX)=4E20H
IMUL  BX                ;(DX:AX)=(AX)*(BX)
```

以上指令的执行结果(DX)＝017FH,(AX)＝4D00H,且(CF)＝(OF)＝1。实际上以上指令完成带符号数(＋1256)和(＋20000)的乘法运算,得到乘积为(＋25120000)。由于此时DX中结果的高半部分包含着乘积的有效数字,故状态标志位(CF)＝(OF)＝1。

所谓乘积的高半部分仅仅是低半部分符号位的扩展,是指当乘积为正值时,其符号位为零,则AH或DX的高半部分为8位全零或16位全零。当乘积是负值时,其符号位为1,则AH或DX的高半部分为8位全1或16位全1。这种情况表示所得乘积的绝对值比较小,其有效数位仅仅包含在低半部分中。

4. 除法指令

8086/8088CPU执行除法时规定:除数只能是被除数的一半字长。当被除数为16位时,除数应为8位;被除数为32位时,除数应为16位。并规定：

① 当被除数为16位,应存放于AX中。除数为8位,可存放在寄存器/存储器中。而得到的8位商放在AL中,8位余数放在AH中。如图4-12(a)所示。

② 当被除数为32位,应存放于DX和AX中。除数为16位,可存放在寄存器/存储器中。而得到的16位商放在AX中,16位余数放在DX中。如图4-12(b)所示。

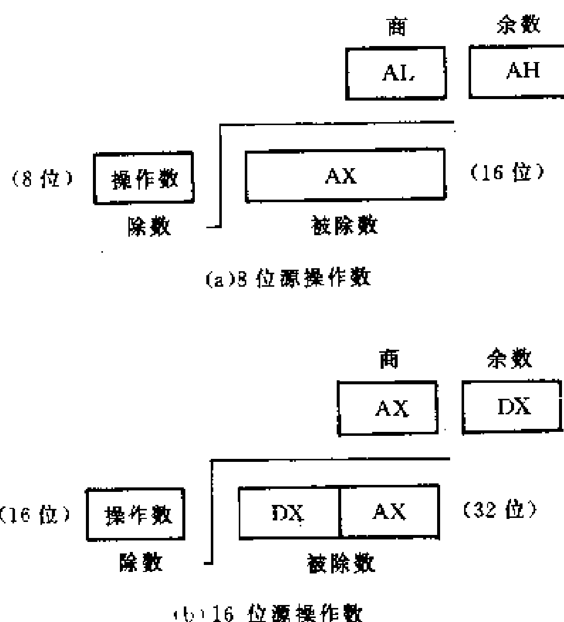


图4-12　除法运算的操作数和运算结果

8086/8088指令系统中有两条除法指令,它们是无符号数除法指令和带符号数的除法指令。

(1) 无符号数除法指令

指令格式及操作：

```
DIV  src      ;(AL)←(AX)/(src)的商          (字节除法)
```

;(AH)←(AX)/(src)的余数
;(AX)←(DX:AX)/(src)的商　　　　(字除法)
;(DX)←(DX:AX)/(src)的余数

即字节除法中,AX除以src,被除数为16位,除数为8位。执行DIV指令后,商在AL,余数在AH中;字除法中,DX、AX除以src,被除数为32位,除数为16位。除的结果,商在AX,余数在DX中。

执行DIV指令时,如果除数为0,或字节除法时AL寄存器中的商大于FFH,或字除法时AX寄存器中的商大于FFFFH,则CPU立即自动产生一个类型号为0的内部中断。有关中断的概念将在本书第八章中详细讨论。

DIV指令使状态标志位如SF、ZF、AF、PF、CF和OF的值不确定。

在DIV指令中,一个操作数(被除数)隐含在累加器AX(字节除法)或DX、AX(字除法)中,另一个操作数src(除数),必须是寄存器或存储器操作数。两个操作数被作为无符号数对待。例如:

```
DIV  BL                        ;AX除以BL
DIV  CX                        ;DX:AX除以CX
DIV  BYTE PTR DATA             ;AX除以存储器(8位)
DIV  WORD PTR[DI+BX]           ;DX:AX除以存储器(16位)
```

下面几条指令将DX:AX中的一个32位无符号数除以CX中的一个16位无符号数。

```
MOV  AX,0F05H                  ;(AX)=0F05H
MOV  DX,068AH                  ;(DX)=068AH
MOV  CX,08E9H                  ;(CX)=08E9H
DIV  CX                        ;(AX)=BBE1H,(DX)=073CH
```

执行结果为:068A0F05H÷08E9H=BBE1H…073CH

除法指令规定了必须用一个16位数除以一个8位数,或用一个32位数除以一个16位数,而不允许两个字长相等的操作数相除。如果被除数和除数的字长相等,可以在用DIV指令进行无符号数除法之前将被除数的高位扩展8个零或16个零。

(2)带符号除法指令

指令格式及操作:

```
IDIV  src     ;(AL)←(AX)/(src)的商          (字节除法)
              ;(AH)←(AX)/(src)的余数
              ;(AX)←(DX:AX)/(src)的商       (字除法)
              ;(DX)←(DX:AX)/(src)的余数
```

执行IDIV指令时,如除数为0,或执行字节除法时AL寄存器中的商超出(−128～+127)的范围,或字除法时AX寄存器中的商超出(−32768～+32767)的范围,则自动产生一个类型为0的中断。另外,IDIV指令对状态标志位的影响以及指令中操作数的类型与DIV指令相同。

例如:如果被除数和除数字长相等,则在用IDIV指令进行带符号数除法之前,必须先用符号扩展指令CBW或CWD将被除数的符号位扩展,使之成为16位数或32位数。关于CBW和CWD指令,本节后面将进行介绍。

IDIV指令对非整数商舍去尾数,而余数的符号总是与被除数的符号相同。下面是一个字

长相等的带符号数除法的例子。

```
MOV   AX,-2000       ;(AX)=-2000
CWD                  ;将 AX 中的 16 位数扩展成为 32 位,结果在 DX:AX
MOV   BX,-421        ;(BX)=-421
IDIV  BX             ;(AX)=4(商),(DX)=-316(余数)
```

除法结果得到商为 4,余数为(-316),余数的符号与被除数相同。

5. 符号扩展指令

在前面介绍的各种二进制算术运算指令中,两个操作数的字长应该符合规定的关系。例如,在加法、减法和乘法运算中,两个操作数的字长必须相等。在除法指令中,被除数必须是除数的双倍字长。因此,有时需要将一个 8 位数扩展成为 16 位,或者将一个 16 位数扩展成为 32 位。

对于无符号数,扩展字长比较简单,只需添上足够个数的零即可。例如,以下两条指令将 AL 中的一个 8 位无符号数扩展成为 16 位,存放在 AX 中。

```
MOV   AL,0FBH        ;(AL)=11111011B
XOR   AH,AH          ;(AH)=00000000B
```

但是,对于带符号数,扩展字长时正数与负数的处理方法不同。正数的符号位为零,而负数的符号位为 1,因此,扩展字长时,应分别在高位添上相应的符号位,这样才能保证原数据的大小和符号不变。符号扩展指令就是用来对带符号数字长的扩展。

(1)字节扩展指令

指令格式及操作:

```
CBW                  ;如果 (AL)<80H,则(AH)←00H,否则(AH)←FFH
```

CBW 指令将一个字节(8 位)按其符号扩展成为字(16 位)。它是一个隐含操作数的指令,隐含的操作数为寄存器 AL 和 AH 中的值。CBW 指令对状态标志位没有影响。

观察下面两组指令,由于初始时 AL 寄存器中内容的符号位不同,因而执行 CBW 指令后 AH 中的结果也不同。

```
(a) MOV   AL,4FH     ;(AL)=01001111B
    CBW              ;(AH)=00000000B
(b) MOV   AL,0F4H    ;(AL)=11110100B
    CBW              ;(AH)=11111111B
```

(2)字扩展指令

指令格式及操作:

```
CWD                  ;如果 (AX)<8000H,则(DX)←0000H,否则(DX)←FFFFH
```

CWD 指令将一个字(16 位)按其符号扩展成为双字(32 位)。它也是一个隐含操作数的指令,隐含的操作数为寄存器 AX 和 DX 中的值。CWD 指令与 CBW 一样,对状态标志位没有影响。

CBW 和 CWD 指令在带符号数的乘法(IMUL)和除法(IDIV)运算中十分有用,常常字节或字的运算之前,将 AL 和 AX 中数据的符号位进行扩展。例如:

```
MOV   AL,MUL—BYTE    ;(AL)←8 位被乘数(带符号数)
CBW                  ;扩展成为 16 位带符号数,在 AX 中
IMUL  RSRC-WORD      ;两个 16 位带符号数相乘,结果在 DX:AX 中
```

(四)十进制数(BCD 码)运算指令

以上我们介绍的是二进制数的算术运算。二进制数在计算机上进行运算是非常简单的。但是,通常人们习惯于用十进制数。在计算机中十进制数是用 BCD 码来表示的。BCD 码有两类:一类叫压缩型 BCD 码,一类叫非压缩型 BCD 码。用 BCD 码进行加、减、乘、除运算,通常采用两种方法:一种是在指令系统中设置一套专用于 BCD 码运算的指令;另一种是利用二进制数的运算指令算出结果,然后再用专门的指令对结果进行修正(调整),使之转变为正确的 BCD 码表示的结果。8086/8088 指令系统所采用的是后一种方法。

在进行十进制数算术运算时,应分两步进行:

① 先按二进制数运算规则进行运算,得到中间结果。

② 再用十进制调整指令对中间结果进行修正,得到正确的结果。

下面通过几个例子说明 BCD 码运算为什么要调整以及怎样进行调整。

34+23=57		0011 0100	23 的 BCD 码
	+	0010 0011	23 的 BCD 码
		0101 0111	57 的 BCD 码

结果正确,这时调整指令不需要做什么。

48+29=77		0100 1000	48 的 BCD 码
	+	0010 1001	29 的 BCD 码
		0111 0001	71 的 BCD 码

结果不正确,因为在进行二进制加法运算时,低 4 位向高 4 位有一个进位,这个进位是按十六进制进行的,即低 4 位逢十六才进一,而十进制数应是逢十进一。因此,比正确结果少 6,这时,调整指令应在低 4 位上加 6。即:

	0100 1000	
+	0010 1001	
	0111 0001	中间结果 AF=1
+	0000 0110	加 06H 调整
	0111 0111	正确结果

加法运算后,低 4 位若向高 4 位有进位(即 AF=1)时,调整指令应做加 06H 处理。

57+46=103		0101 0111	
	+	0100 0110	
		1001 1101	中间结果
	+	0000 0110	
		1010 0011	中间结果
	+	0110 0000	
	CF←1	0000 0011	正确结果　CF=1

加法运算后,低 4 位>9 时,调整指令需做加 06H 处理;高 4 位>9 时,调整指令需做加 60H 处理。

```
72+91=163          0111 0010
               +   1001 0001
             CF←1  0000 0011      中间结果 CF=1

               +   0110 0000
                   0110 0011      正确结果
```

加法运算后，当 CF=1(有进位产生)时，调整指令应做加 60H 处理。

前面我们已经详细地介绍了二进制数的算术运算指令，下面主要介绍十进制数(BCD 码)的调整指令。

1. 十进制加法的调整指令

根据 BCD 码的种类，对 BCD 码加法进行十进制调整的指令有两条 AAA 和 DAA。

(1) 非压缩型 BCD 码调整指令

指令格式：

AAA

AAA 也称为加法的 ASCII 调整指令。指令后面不写操作数，但实际上隐含累加器操作数 AL 和 AH。指令的操作为：

```
如果    (AL)∧0FH>9,或(AF)=1
则      (AL)←(AL)+06H
        (AH)←(AH)+1
        (AF)←1
        (CF)←(AF)
        (AL)←((AL)∧0FH)
否则    (AL)←((AL)∧0FH)
```

由上可见，指令将影响 AF 和 CF 标志，但状态标志位 SF、ZF、PF 和 OF 的状态不确定。

在用 AAA 指令调整以前，先用指令 ADD(多字节加法时用 ADC)进行 8 位数的加法运算，相加结果放在 AL 中，用 AAA 指令调整后，非压缩型 BCD 码结果的低位在 AL 寄存器，高位在 AH 寄存器。例如，要求计算两个十进制数之和，7+8=?。可以先将被加数 7、加数 8 以非压缩型 BCD 码的形式分别存放在寄存器 AL 和 BL 中，且令 AH=0，然后进行加法，再用 AAA 指令调整。可用以下指令实现：

```
MOV   AX,0007H       ;(AL)=07H,(AH)=00H
MOV   BL,08H         ;(BL)=08H
ADD   AL,BL          ;(AL)=0FH
AAA                  ;(AL)=05H,(AH)=01H,(CF)=(AF)=1
```

以上指令的运行结果为 7+8=15，所得之和也以非压缩型 BCD 码的形式存放，个位在 AL，十位在 AH。

例 4.4 计算 4609+3875=?

本例要求实现十进制多位数的加法，假设被加数的每一位数都以 ASCII 码形式存放在内存中，低位在前，高位在后。另外留出 4 个存储单元，以便存放相加所得的结果，如图 4-13 所示。程序的流程图见图 4-14。

程序如下：

```
LEA     SI,STRING1      ;(SI)←被加数地址指针
LEA     BX,STRING2      ;(BX)←加数地址指针
LEA     DI,SUM          ;(DI)←结果地址指针
MOV     CX,4            ;(CX)←循环次数
CLC                     ;清进位标志 CF
NEXT:   MOV   AL,[SI]   ;取一个字节被加数
        ADC   AL,[BX]   ;与加数相加
        AAA             ;ASCII 调整
        MOV   [DI],AL   ;送存
        INC   SI        ;SI 加 1
        INC   BX        ;BX 加 1
        INC   DI        ;DI 加 1
        DEC   CX        ;循环次数减 1
        JNZ   NEXT      ;如不为零,转 NEXT
        HLT             ;停止
```

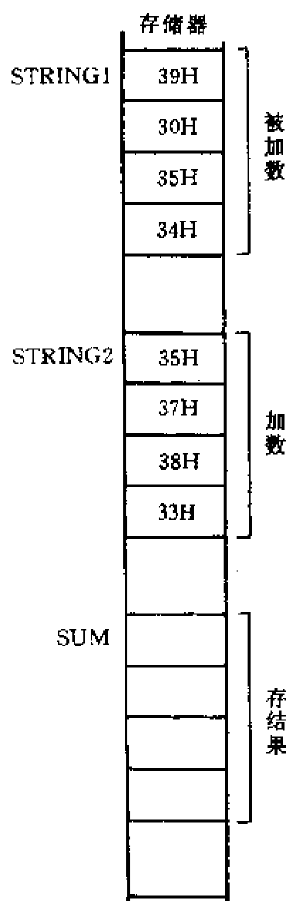


图 4-13　数据存放情况

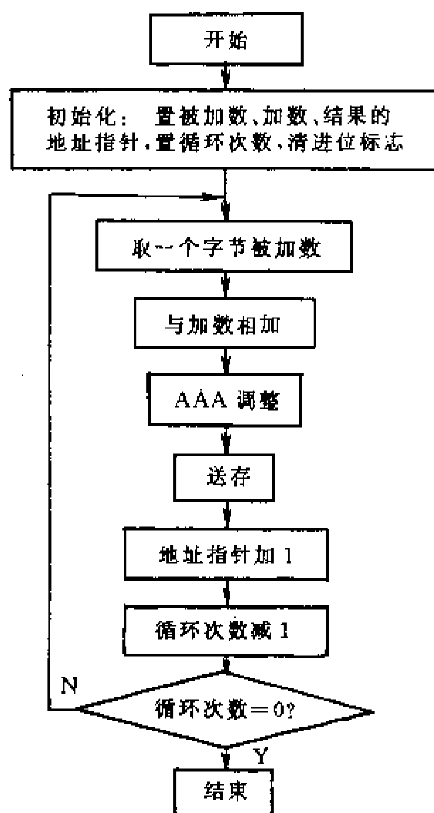


图 4-14 例 4.4 程序流程图

(2)压缩型 BCD 码调整指令

指令格式：

DAA

DAA 指令同样不带操作数，实际上隐含寄存器操作数 AL。指令的操作为：

如果　((AL)∧0FH)>9 或 (AF)=1

则　(AL)←(AL)+06H

　　(AF)←1

如果　(AL)>9FH 或 (CF)=1

则　(AL)←(AL)+60H

　　(CF)←1

与 AAA 指令不同，DAA 只对 AL 中的内容进行调整，任何时候都不会改变 AH 的内容。另外，DAA 指令将影响状态标志位，如 SF、ZF、AF、PF、CF，但不影响 OF。例如，要求计算两个 2 位的十进制数之和，68+59=?。调整之前，也应先用 ADD 或 ADC 指令进行 8 位数加法运算，相加结果放在 AL 中，然后用 DAA 指令进行调整。可用以下指令实现：

```
MOV   AL,68H              ;(AL)=68H
MOV   BL,59H              ;(BL)=59H
ADD   AL,BL               ;(AL)=C1H,(AF)=1
DAA                       ;(AL)=27H,(CF)=1
```

如果要求相加两个位数更多的十进制数，则也应编写一个循环程序，并采用 ADC 指令，在开始循环之前要清进位标志 CF(参阅例 4.4)。但采用压缩型 BCD 码时每次可以相加两位十进制数。例如，相加两个 8 位十进制数时只需循环 4 次。

为了掌握 DAA 指令与 AAA 指令的区别，现在再来做前面已经作过的简单计算，即

7+8=?

不过这一次编程时不用 AAA 指令，而改用 DAA 指令调整，看看结果有什么不同。

```
MOV   AX,0007H     ;(AL)=07H,(AH)=00H
MOV   BL,08H       ;(BL)=08H
ADD   AL,BL        ;(AL)=0FH
DAA                ;(AL)=15H,(AH)=00H,(AF)=1,(CF)=0
```

可见，现在 7 加 8 所得之和以压缩型 BCD 码的形式存放在 AL 寄存器中，而 AH 的内容不变。

2. 十进制减法的调整指令

同加法一样，对 BCD 码减法进行十进制调整的指令也有两条 AAS 和 DAS。

(1)非压缩型 BCD 码调整指令

指令格式：

AAS

AAS 也称为减法的 ASCII 码的调整指令。隐含寄存器操作数为 AL 和 AH。

AAS 指令在减法运算时，对非压缩型 BCD 码进行调整，以得到正确的结果。AAS 指令的操作为：

如果　(AL)∧0FH>9 或 (AF)=1

则　(AL)←(AL)−06H

　　(AH)←(AH)−1

　　(AF)←1

　　(CF)←(AF)

(AL)←((AL)∧0FH)

否则　(AL)←((AL)∧0FH)

可见,AAS 指令将影响状态标志位 AF 和 CF,但 SF、ZF、PF 和 OF 状态标志位不确定。

例如想要进行两位十进制数的减法运算:13－4＝?，可先将被减数和减数以非压缩型 BCD 码的形式分别存放在 AH(被减数的十位)、AL(被减数的个位)和 BL(减数)中，然后用 SUB 指令进行减法,再用 AAS 指令进行调整。可用以下指令实现:

```
MOV  AX,0103H        ;(AH)=01H,(AL)=03H
MOV  BL,04H          ;(BL)=04H
SUB  AL,BL           ;(AL)=03H-04H=FFH
AAS                  ;(AL)=09H,(AH)=0
```

以上指令的执行结果为 13－4＝9,此结果仍以非压缩型 BCD 码的形式存放,个位在 AL 寄存器,十位在 AH 寄存器。

(2)压缩型 BCD 码调整指令

指令格式:

DAS

DAS 指令对减法进行十进制调整,指令隐含寄存器操作数 AL。

在减法运算时,DAS 指令对压缩型 BCD 码进行调整,其操作为:

如果　((AL)∧0FH)>9 或 (AF)=1

则　(AL)←(AL)－06H

　　(AF)←1

如果 (AL)>9FH 或 (CF)=1

则　(AL)←(AL)－60H

　　(CF)←1

与 DAA 指令类似,DAS 指令也只对 AL 寄存器中的内容进行调整,而不改变 AH 的内容。DAS 指令也将影响状态标志位 SF、ZF、AF、PF、CF,但不影响 OF。

例如要求完成以下十进制数的减法运算:83－38＝?现在采用压缩型 BCD 码的形式来存放原始数据,则以上减法运算可用下列几条指令实现:

```
MOV  AL,83H          ;(AL)=83H
MOV  BL,38H          ;(BL)=38H
SUB  AL,BL           ;(AL)=4BH
DAS                  ;(AL)=45H
```

3. 十进制乘除法的调整指令

对于十进制数的乘除法运算,8086/8088 指令系统只提供了非压缩型 BCD 码的调整指令,而没有提供压缩型 BCD 码的调整指令。因此,8086/8088 CPU 不能直接进行压缩型 BCD 码的乘除法运算。

非压缩型 BCD 码的乘除法与加减法不同,加减法可以直接用 ASCII 码参加运算,而不管其高位上有无数字,只要在加减指令后用一条非压缩型 BCD 码的调整指令就能得到正确结果。而乘除法要求参加运算的两个数必须是高 4 位为 0 的非压缩型 BCD 码，低 4 位为一个十进制数。也就是说,如果用 ASCII 码进行非压缩型 BCD 码乘除法运算的话,在乘除法运算之前,必须将高 4 位清零。

(1)非压缩型 BCD 码的乘法调整指令

指令格式：

AAM

AAM 指令也是一个隐含了寄存器操作数 AL 和 AH 的指令。

在乘法运算时，调整之前，先用 MUL 指令将两个真正的非压缩型的 BCD 码相乘，结果放在 AX 中。然后用 AAM 指令对 AL 寄存器进行调整，于是在 AX 中就可得到正确的非压缩型 BCD 码的结果，其乘积的高位在 AH 中，乘积的低位在 AL 中。AAM 指令的操作为：

(AH)←(AL)/0AH 的商　　　　即 AL 除以 10，商送 AH

(AL)←(AL)/0AH 的余数　　　即 AL 除以 10，余数送 AL

AAM 指令的操作实质上是将 AL 寄存器中的二进制数转换成为非压缩型的 BCD 码，十位存放在 AH 寄存器，个位存放在 AL 寄存器。

AAM 指令执行以后，将根据 AL 中的结果影响状态标志位 SF、Z 和 PF，但 AF、CF 和 OF 的值不定。

例如要求进行以下十进制乘法运算：7×9＝? 可编程序段如下：

```
MOV   AL,07H     ;(AL)=07H
MOV   BL,09H     ;(BL)=09H
MUL   BL         ;(AX)=07H×09H=003FH
AAM              ;(AH)=06H,(AL)=03H,(SF)=0,(ZF)=0,(PF)=1
```

已知 7×9＝63。以上指令执行以后，十进制乘积也是以非压缩型 BCD 码的形式存放在 AX 中。由于(AL)＝03H(即 00000011B)，故决定了(SF)＝0，(ZF)＝0，(PF)＝1。

(2)非压缩型 BCD 码的除法调整指令

指令格式：

AAD

AAD 指令也是一个隐含了寄存器操作数 AL 和 AH 的指令。它是对非压缩型 BCD 码进行调整，其操作为：

(AL)←(AH)×0AH＋(AL)

(AH)←0

即将 AH 寄存器的内容乘以 10 并加上 AL 寄存器的内容，结果送回 AL，同时将零送 AH。以上操作实质上是将 AX 寄存器中非压缩型 BCD 码转换成为真正的二进制数，并存放在 AL 寄存器中。

执行 AAD 指令以后，将根据 AL 中的结果影响状态标志位 SF、ZF 和 PF，但其余几个状态标志位如 AF、CF 和 OF 的值则不确定。

AAD 指令的用法与其他非压缩型 BCD 码调整指令(如 AAA、AAS、AAM)有所不同。AAD 指令不是在除法之后，而是在除法之前进行调整，然后用 DIV 指令进行除法，所得之商还需用 AAM 指令进行调整，方可得到正确的非压缩型 BCD 码的结果。

例如想要进行以下十进制除法运算：73÷2＝?可先将被除数和除数以非压缩型 BCD 码的形式分别存放在 AX 和 BL 寄存器中，被除数的十位在 AH，个位在 AL，除数在 BL。先用 AAD 指令对 AX 中的被除数进行调整，之后进行除法运算，并对商进行再调整。可编成如下程序段：

```
MOV   AX,0703H          ;(AH)=07H,(AL)=03H
MOV   BL,02H            ;(BL)=02H
```

```
AAD                         ;(AL)=29H(即十进制数 73)
DIV   BL                    ;(AL)=24H(商),(AH)=01H(余数)
AAM                         ;(AH)=03H,(AL)=06H
```

已知 73÷2=36…1。以上几条指令执行的结果为,在 AX 中得到非压缩型 BCD 码的商,但余数被丢失。如果需要保留余数,则应在 DIV 指令之后,用 AAM 指令调整之前,将余数暂存到一个寄存器。如果有必要,还应设法对余数也进行调整。

三、位操作指令

位操作指令是对 8 位或 16 位的寄存器或存储单元中的内容按位进行操作。这一类指令包括逻辑运算指令、移位指令和循环移位指令等三组。

(一)逻辑运算指令

8086/8088 指令系统的逻辑运算指令有 AND(逻辑"与")、TEST(测试)、OR (逻辑"或")、XOR(逻辑"异或")和 NOT(逻辑"非")五条指令,这些指令对操作数中的各个位分别进行布尔运算。各种逻辑运算的结果如表 4-4 所示。

表 4-4　逻辑运算返回的值

X	Y	X AND Y	X OR Y	X XOR Y	NOT X
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

以上五条逻辑运算指令中,唯有 NOT 指令对状态标志位不产生影响,其余四条指令(即 AND、TEST、OR 和 XOR)对状态标志位均有影响。这些指令将根据各自逻辑运算的结果影响 SF、ZF 和 PF 状态标志位,同时将 CF 和 OF 置"0",但使 AF 的值不确定。

1.逻辑"与"指令

指令格式及操作:

```
AND   dst,src               ;(dst)←(dst)∧(src)
```

AND 指令将目标操作数和源操作数按位进行逻辑"与"运算,并将结果送回目标操作数。

目标操作数可以是寄存器或存储器,源操作数可以是立即数、寄存器或存储器。但是指令的两个操作数不能同时是存储器,即不能将两个存储器的内容进行逻辑"与"操作。AND 指令操作对象的类型可以是字节,也可以是字。例如:

```
AND   AL,00001111H          ;寄存器"与"立即数
AND   CX,DI                 ;寄存器"与"寄存器
AND   SI,MEM—NAME           ;寄存器"与"存储器
AND   ALPHA[DI],AX          ;存储器"与"寄存器
AND   [BX][SI],0FFFEH       ;存储器"与"立即数
```

AND 指令可以用于屏蔽某些不关心的位,而保留另一些感兴趣的位。为了做到这一点,只需将屏蔽的位和"0"进行逻辑"与",而将要求保留的位和"1"进行逻辑"与"即可。

例如 AND　AL,0FH 指令将 AL 寄存器中的内容屏蔽高 4 位,保留低 4 位。该指令可将数字 0~9 的 ASCII 码转换成相应的非压缩型 BCD 码。例如:

```
MOV   AL,'6'                ;(AL)=00110110B
```

```
AND  AL,0FH                  ;(AL)=00000110B
```

利用 AND AL,11011111B 指令可以将 AL 中的英文字母(用 ASCII 码表示)转换成为大写字母。如果字母原来已经是大写,则以上 AND 指令不起作用,因为大写字母 ASCII 码的第 5 位总是 0;如果原来是小写字母,则将其第 5 位置"0",转换成为相应的大写字母。大写和小写英文字母 ASCII 码的对比如下表 4-5 所示。

表 4-5 大写和小写英文字母 ASCII 码的对比

大 写 字 母	小 写 字 母
'A'=41H=0100 0001B	'a'=61H=0110 0001B
'B'=42H=0100 0010B	'b'=62H=0110 0010B
'Z'=5AH=0101 1010B	'z'=7AH=0111 1010B

以下几条指令判断从键盘输入的字符是否为"Y",但对键入的字符大写或小写不加区别,同样对待。

```
    MOV  AH,7              ;接收由键盘输入的字符
    INT  21H               ;字符的 ASCII 码存 AL
    AND  AL,11011111B      ;屏蔽第 5 位,转换为大写字母
    CMP  AL,'Y'            ;字符是否为"Y"?
    JE   YES               ;如是,转到 YES
    :                      ;否则,…
YES:  …
```

以上程序中的前两条指令涉及 DOS 的功能调用,将在本书第五章第六节中详细进行讨论。

2. 测试指令

指令格式及操作:

TEST dst,src ;(dst)∧(src)

TEST 指令的操作实质上与 AND 指令相同,即把目标操作数和源操作数进行逻辑"与"。二者的区别在于 TEST 指令不把逻辑运算的结果送回目标操作数,因此两个操作数的内容均保持不变,即目标操作数将不被破坏。逻辑"与"的结果反映在状态标志位上,例如,"与"的结果最高位是"0"还是"1",结果是否为全"0",结果中"1 "的个数是奇数还是偶数等,分别由 SF、ZF 和 PF 状态标志位体现。和 AND 指令一样,TEST 指令总是将 CF 和 OF 清零,但使 AF 的值不确定。

TEST 指令的例子如下:

```
TEST  BH,7                 ;寄存器"与"立即数(结果不回送,下同)
TEST  SI,BP                ;寄存器"与"寄存器
TEST  [SI],CH              ;存储器"与"寄存器
TEST  [BX][DI],6ACEH       ;存储器"与"立即数
```

TEST 指令常常用于位测试,它与条件转移指令一起,共同完成对特定位状态的判断,并实现相应的程序转移。这样的作用与比较指令 CMP 有些类似,不过 TEST 指令只比较某一个指定的位,而 CMP 指令比较整个操作数(字节或字)。例如以下几条指令判断一个端口地址为 PORT 的外设端口输入的数据,若输入数据的第 1、3、5 位中的任一位不等于零,则转移到 NEXT。

```
IN  AL,PORT
```

```
TEST  AL,00101010B
JNZ  NEXT
NEXT: …
```

3. 逻辑"或"指令

指令格式及操作:

```
OR   dst,src                  ;(dst)←(dst)∨(src)
```

OR 指令将目的操作和源操作数按位进行逻辑"或"运算,并将结果送回目标操作数。

OR 指令操作数的类型与 AND 指令相同,即目标操作数可以是寄存器或存储器,源操作数可以是立即数、寄存器或存储器。但两个操作数不能同时都是存储器。例如:

```
OR   BL,0F6H                  ;寄存器"或"立即数
OR   AH,BL                    ;寄存器"或"寄存器
OR   CL,BETA[BX][DI]          ;寄存器"或"存储器
OR   GAMMA[SI],DX             ;存储器"或"寄存器
OR   MEM—BYTE,80H             ;存储器"或"立即数
```

OR 指令的一个常见的用途是将寄存器或存储器中某些特定的位设置成"1",而不管这些位原来的状态如何,同时使其余位保持原来的状态不变。为此,应将需置"1"的位和"1"进行逻辑"或",而将要求保持不变的位和"0"进行逻辑"或"。例如,以下指令可将 AH 寄存器及 AL 寄存器的最高位同时置"1",而 AX 中的其余位保持不变:

```
OR   AX,8080H                 ;(AX)∨(1000000010000000B)
```

AND 指令和 OR 指令有一个共同的特性:如果将一个寄存器的内容和该寄存器本身进行逻辑"与"操作或者逻辑"或"操作,则寄存器原来的内容不会改变,但寄存器中的内容将影响 SF、ZF 和 PF 状态标志位,且将 OF 和 CF 清零。

利用这个特性,可以在数据传送指令之后,使该数据影响标志,然后可以判断数据的正负。又如,以下几条指令判断数据是否为零。

```
MOV  AX,DATA                  ;(AX)←DATA
OR   AX,AX                    ;影响标志(用 AND AX,AX 指令亦可)
JZ   ZERO                     ;如为零,转移到 ZERO
 ⋮                            ;否则,…
ZERO:                          …
```

在以上程序中,如果没有 OR AX,AX(或 AND AX,AX)指令,则不能紧跟着进行条件判断和程序转移,因为 MOV 指令不影响标志。当然采用 CMP AX,0 指令代替上述 AND 或 OR 指令也可以得到同样的效果,但这条比较指令字节较多,且执行速度较慢。逻辑运算指令和比较指令的字节数和时钟数如下:

```
AND  AX,AX                    ;2 个字节,3 个时钟数
OR   AX,AX                    ;2 个字节,3 个时钟数
CMP  AX,0                     ;3 个字节,4 个时钟数
```

4. 逻辑"异或"指令

指令格式及操作:

```
XOR  dst,sre                  ;(dst)←(dst)⊕(src)
```

XOR 指令将目标操作数和源操作数按位进行逻辑"异或"运算,并将结果送回目标操作

数。

XOR 指令操作数的类型和 AND、OR 指令均相同，此处不再赘述，请看下面的例子：

```
XOR   DI,23F6H              ;寄存器“异或”立即数
XOR   SI,DX                 ;寄存器“异或”寄存器
XOR   CL,BUFFER             ;寄存器“异或”存储器
XOR   MEM[BX],AX            ;存储器“异或”寄存器
XOR   TABLE[BP][SI],3DH     ;存储器“异或”立即数
```

XOR 指令的一个用途是将寄存器或存储器中某些特定的位“求反”，而使其余位保持不变。为此，可将欲“求反”的位和“1”进行“异或”，而将要求保持不变的位和“0”进行“异或”。例如，若要使 AL 寄存器中的第 1、3、5、7 位求反，第 0、2、4、6 位保持不变，则只需将 AL 和 10101010B(即 0AAH)“异或”即可。

```
MOV   AL,0FH                ;(AL)=00001111B
XOR   AL,0AAH               ;(AL)=10100101B(0A5H)
```

XOR 指令的另一个用途是将寄存器的内容清零，例如：

```
XOR   AX,AX                 ;AX 清零
XOR   CX,CX                 ;CX 清零
```

而且，上述指令和 AND、OR 等指令一样，也将进位标志 CF 清零。

XOR 指令的这种特性在多字节的累加程序中十分有用，它可以在循环程序开始前的初始化工作中将一个用作累加器的寄存器清零，并使进位标志 CF 清零。

例 4.5　从偏移地址 TABLE 开始的内存区中，存放着 100 个字节的十六进制数，要求将这些数进行累加，并将累加和的低位存 SUM 单元，高位存 SUM+1 单元。程序如下：

```
         LEA   BX,TABLE     ;(BX)←数据表地址指针
         MOV   CL,100       ;(CL)←数据块长度
         XOR   AX,AX        ;清 AL、AH
LOOPER:  ADD   AL,[BX]      ;加一个数到 AL
         JNC   GOON         ;如(CF)=0,转移到 GOON
         INC   AH           ;否则,AH 加 1
GOON:    INC   BX           ;地址指针加 1
         DEC   CL           ;计数值减 1
         JNZ   LOOPER       ;如(CL)≠0,转移到 LOOPER
         MOV   SUM,AX       ;否则,(SUM)←(AL),(SUM+1)←(AH)
         HLT                ;停止
```

5. 逻辑“非”运算

指令格式及操作：

```
NOT   dst                   ;(dst)←FFFFH-(dst)(字求反)
```

NOT 指令的操作数可以是 8 位或 16 位的寄存器或存储器，但不能对一个立即数执行逻辑“非”操作。以下是 NOT 指令的几个例子：

```
NOT   AH                    ;8 位寄存器求反
NOT   CX                    ;16 位寄存器求反
NOT   BYTE PTR[BP]          ;8 位存储器求反
```

NOT　WORD PTR COUNT　　;16 位存储器求反

(二)移位指令

8086/8088 指令系统的移位指令包括逻辑左移 SHL、算术左移 SAL、逻辑右移 SHR、算术右移 SAR 等指令，其中 SHL 和 SAL 指令的操作完全相同。移位指令的操作对象可以是一个 8 位或 16 位的寄存器或存储器，移位操作可以是向左或向右移一位，也可以移多位。当要求移多位时，指令规定移动位数必须放在 CL 寄存器中，即指令中规定的移位次数不允许是 1 以外的常数或 CL 以外的寄存器。移位指令都影响状态标志位，但影响的方式各条指令不尽相同。

1. 逻辑左移/算术左移指令

指令格式：

```
    SHL  dst,1/SAL dst,1
或  SHL  dst,CL/SAL dst,CL
```

这两条指令的操作是将目标操作数顺序向左移 1 位或移 CL 寄存器中指定的位数。左移 1 位时，操作数的最高位移入进位标志 CF，最低位补 0，其操作如图 4-15 所示。



图 4-15　SHL/SAL 指令操作示意图

SHL/SAL 指令将影响 CF 和 OF 两个状态标志位，如果移位次数等于 1，且移位以后目标操作数新的最高位与 CF 不相等，则溢出标志 OF=1，否则 OF=0。因此 OF 的值表示移位操作是否改变了符号位。如果移位次数不等于 1，则 OF 的值不确定。指令对其他状态标志位没有影响。以下是 SHL/SAL 指令的几个例子：

```
SHL  AH,1                  ;寄存器左移 1 位
SAL  SI,CL                 ;寄存器左移(CL)位
SAL  WORD PTR[BX+5],1      ;存储器左移 1 位
SHL  BYTE PTR DATA,CL      ;存储器左移(CL)位
```

例 4.6　将一个 16 位无符号数乘以 10。该数原来存放在以 FACTOR 为首地址的两个连续的存储单元中(低位在前，高位在后)。

因为 FACTOR×10=(FACTOR×8)+(FACTOR×2)，故可用左移指令实现以上乘法运算。编程如下：

```
MOV  AX,FACTOR      ;(AX)←被乘数
SHL  AX,1           ;(AX)=FACTOR×2
MOV  BX,AX          ;暂存 BX
SHL  AX,1           ;(AX)=FACTOR×4
SHL  AX,1           ;(AX)=FACTOR×8
ADD  AX,BX          ;(AX)=FACTOR×10
HLT
```

以上程序的执行时间大约需 26 个周期。如用乘法指令编程，执行时间将超过 130 个时钟数。

2. 逻辑右移指令

指令格式：

```
SHR  dst,1/CL
```

SHR 指令的操作是将目标操作数顺序向右移 1 位或右移由 CL 寄存器指定的位数。逻辑

右移一位时，操作数的最低位移到进位标志CF，最高位补0，指令操作如图4-16所示。

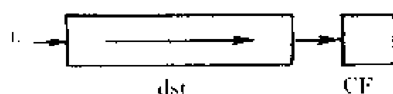


图4-16 SHR指令操作示意图

SHR指令也将影响CF和OF状态标志位。如果移位次数等于1，且移位以后新的最高位与次高位不相等，则OF=1，否则OF=0。实质上此时OF的值仍然表示符号位在移位前后是否改变。如果移位次数不等于1，则OF的值不定。

下面举出SHR指令的几个例子。

```
SHR  BL,1                       ;寄存器逻辑右移1位
SHR  AX,CL                      ;寄存器逻辑右移(CL)位
SHR  BYTE PTR[DI+BP],1          ;存储器逻辑右移1位
SHR  WORD PTR BLOCK,CL          ;存储器逻辑右移(CL)位
```

例4.7 将一个16位无符号数除以512。该数原来存放在以DIVIDAND为首地址的两个连续的存储单元中。

因为DIVIDAND÷512=(DIVIDAND÷2)÷256，因此可用逻辑右移指令完成上述除法运算。

编程如下：

```
MOV   AX,DIVIDAND       ;(AX)←被乘数
SHR   AX,1              ;(AX)=DIVIDAND÷2
XCHG  AL,AH             ;(AL)←→(AH)，相当循环右移8位
CBW                     ;清AX的高8位，(AX)=DIVIDAND÷512
HLT
```

当然，也可以将立即数9传送到CL寄存器，然后用指令SHR AX,CL完成除以512的运算。但是相比之下，上面的程序执行速度更快。

3. 算术右移指令

指令格式：

```
SAR  dst,1/CL
```

SAR指令的操作数与逻辑右移指令SHR有点类似，将目标操作数向右移1位或由CL寄存器指定的位数，操作数的最低位移到进位标志CF。它与SHR指令的主要区别是，算术右移时，最高位保持不变。SAR指令的操作如图4-17所示。



图4-17 SAR指令操作示意图

算术右移指令对状态标志位CF、OF、PF、SF和ZF有影响，但使AF的值不确定。

算术右移指令SAR的例子如下：

```
SAR  AL,1                       ;寄存器算术右移1位
SAR  DI,CL                      ;寄存器算术右移(CL)位
SAR  WORD PTR TABLE[SI],1       ;存储器算术右移1位
```

```
SAR   BYTE PTR STATUS,CL              ;存储器算术右移(CL)位
```

(三)循环移位指令

8086/8088 指令系统有四条循环移位指令,它们是不带进位标志 CF 的左循环移位指令 ROL 和右循环移位指令 ROR(也称小循环),以及带进位标志 CF 的左循环移位指令 RCL 和右循环移位指令 RCR(也称大循环)。

循环移位指令的操作数类型与移位指令相同,可以是 8 位或 16 位的寄存器或存储器。指令中指定的左移或右移的位数也可以是 1 或由 CL 寄存器指定。但不能是 1 以外的常数或 CL 以外的其他寄存器。

所有循环移位指令都只影响进位标志 CF 和溢出标志 OF,但 OF 标志的含义对于左循环移位指令和右循环移位指令有所不同。

1. 循环左移指令

指令格式:

```
ROL dst,1/CL
```

ROL 指令将目标操作数向左循环移动 1 位或 CL 寄存器指定的位数。最高位移到进位标志 CF,同时,最高位移到最低位形成循环,进位标志 CF 不在循环回路之内。其操作如图 4-18 所示。

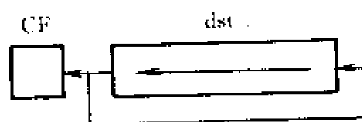


图 4-18　ROL 指令操作示意图

ROL 指令将影响 CF 和 OF 两个状态标志位。如果循环移位次数等于 1,且移位以后目的操作数新的最高位与 CF 不相等,则(OF)=1,否则(OF)=0。因此,OF 的值表示循环移位前后符号位是否有所变化。如果移位次数不等于 1,则 OF 的值不确定。

ROL 指令的例子如下:

```
ROL   BH,1                     ;寄存器循环左移 1 位
ROL   DX,CL                    ;寄存器循环左移(CL)位
ROL   WORD PTR [DI],1          ;存储器循环左位 1 位
ROL   BYTE PTR ALPHA,CL        ;存储器循环左位(CL)位
```

2. 循环右移指令

指令格式:

```
ROR   dst,1/CL
```

ROR 指令将目标操作数向右循环移动 1 位或右移由 CL 寄存器指定的位数。最低位移到进位标志 CF,同时最低位移到最高位,指令的操作可用图 4-19 表示。

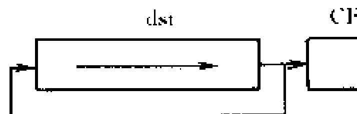


图 4-19　ROR 指令操作示意图

ROR 指令也将影响状态标志位 CF 和 OF。若循环移位次数等于 1 且移位后新的最高位和次高位不等,则(OF)=1,否则(OF)=0。若循环移位次数不等于 1,则 OF 的值不确定。

下面是 ROR 指令的几个例子。

```
ROR   CX,1                     ;寄存器循环右移 1 位
ROR   BH,CL                    ;寄存器循环右移(CL)位
```

```
        ROR   BYTE PTR BETA,1        ;存储器循环右移1位
        ROR   WORD PTR ALPHA,        ;存储器循环右移(CL)位
CL
```

3. 带进位循环左移指令

指令格式:

```
    RCL   dst,1/CL
```

RCL 指令将目标操作数连同进位标志 CF 一起向左循环移动 1 位或由 CL 寄存器指定的位数。最高位移入 CF,而 CF 移入最低位。指令的操作如图 4-20 所示。

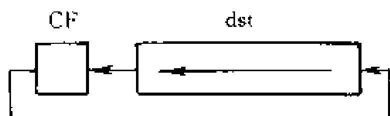


图 4-20 RCL 指令操作示意图

RCL 指令对状态标志位的影响与 ROL 指令相同。

4. 带进位循环右移指令

指令格式:

```
    RCR   dst,1/CL
```

RCR 指令将目标操作数与进位标志 CF 一起向右循环移动 1 位,或由 CL 寄存器指定的位数。最低位移入进位标志 CF,CF 则移入最高位。指令的操作如图 4-21 所示。

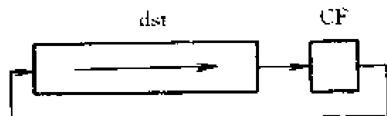


图 4-21 RCR 指令操作示意图

RCR 指令对状态标志位的影响与 ROR 指令相同。

这里介绍的四条循环移位指令与前面讨论过的移位指令有所不同。这四条指令进行循环移位操作之后,操作数中原来各位的信息不会丢失,只是移到了操作数中的其他位或进位标志上,必要时还可以恢复。

利用循环移位指令可以对寄存器或存储器中的任一位进行位测试。例如要求测试 AL 寄存器中第 5 位的状态是"0"还是"1",则可利用以下指令实现:

```
          MOV   CL,5             ;(CL)←移位次数
          ROL   AL,CL            ;(CF)←AL 的第 5 位
          JNC   ZERO             ;若(CF)=0,转 ZERO
          :                      ;否则…
ZERO:     …
```

四、串操作指令

8086/8088 指令系统中有一组十分有用的串操作指令,这些指令的操作对象不只是单个的字节或字,而是内存中地址连续的字节串或字串。在每次基本操作后,能够自动修改地址,为下一次操作做好准备。串操作指令还可以加上重复前缀,此时指令规定的操作将一直重复下去,直到完成预订的重复次数。

串操作指令共有以下五条:串传送指令(MOVS)、串装入指令(LODS)、串送存指令(STOS)、串比较指令(CMPS)和串扫描指令(SCAS)。

上述串操作指令的基本操作各不相同,但都具有以下几个共同特点:

(1)总是用 SI 寄存器寻址源操作数,用 DI 寄存器寻址目标操作数。源操作数常用在现行的数据段,隐含段寄存器 DS,但也允许段超越。目标操作数总是在现行的附加段,隐含段寄存

器 ES,不允许段超越。

(2)每一次操作以后修改地址指针,是增量还是减量决定于方向标志 DF。当(DF)=0 时,地址指针增量,即字节操作时地址指针加 1,字操作时地址指针加 2。当(DF)=1 时,地址指针减量,即字节操作时地址指针减 1,字操作是地址指针减 2。

(3)有的串操作指令可加重复前缀 REP,则指令规定的操作重复进行,重复操作的次数由 CX 寄存器决定。

如果在串操作指令前加上重复前缀 REP,则 CPU 按以下步骤执行:

① 首先检查 CX 寄存器,若(CX)=0,则退出重复串操作指令。

② 指令执行一次字符串基本操作。

③ 根据 DF 标志修改地址指针。

④ CX 减 1(但不改变标志)。

⑤ 转至下一次循环,重复以上步骤。

(4)若串操作指令的基本操作影响零标志 ZF(如 CMPS、SCAS),则可加重复前缀 REPE / REPZ 或 REPNE/REPNZ,此时操作重复进行的条件不仅要求(CX)≠0、而且同时要求 ZF 的值满足重复前缀中的规定(REPE 要求(ZF)=1,REPNE 要求(ZF)=0)

(5)串操作汇编指令的格式可以写上操作数,也可以只在指令助记符后加上字母"B"(字节操作)或"W"(字操作)。加上字母"B"或"W"后,指令助记符后面不允许再写操作数。

下面分别进行介绍。

1. 串传送指令

指令格式:

```
[REP] MOVS [ES:]dst—string,[seg:]src—string
[REP] MOVSB
[REP] MOVSW
```

MOVS 指令也称为字符串传送指令,它将一个字节或字从存储器的某个区域传送到另一个区域,然后根据方向标志 DF 自动修改地址指针。其执行的操作为:

① ((ES):(DI))←((DS):(SI))

② (SI)←(SI)±1,(DI)←(DI)±1 (字节操作)

(SI)←(SI)±2,(DI)←(DI)±2 (字操作)

其中,当方向标志 DF=0 时用"+",当方向标志 DF=1 时用"—"。

串传送指令不影响状态标志位。

以上各种格式中,凡是方括号中内容均表示任选项,即这些项可有可无。例如重复前缀 REP,可以加在串操作指令之前,也可以不加。

在第一种格式中,串操作指令给出源操作数和目标操作数,此时指令执行字节操作还是字操作,决定于这两个操作数定义时的类型。列出源操作数和目标操作数的作用有二。首先,用以说明操作对象的大小(字节或字);其次,明确指出涉及的段寄存器(seg)。指令执行时,实际仍用 SI 和 DI 寄存器寻址操作数。如果在指令中采用 SI 和 DI 来表示操作数,则必须用类型运算符 PTR 说明操作对象的类型。第一种格式的一个重要优点是可以对源字符串进行段重设(目的字符串的段基值只能在 ES,不可进行段重设)。

在第二种和第三种格式中,串操作指令字符的后面加上一个字母"B"或"W",指出操作对象是字节串或字串。但要注意,在这两种情况下,指令后面不允许出现操作数。例如以下指令

都是合法的：

```
REP   MOVS   DATA2,DATA1                 ;操作数类型应预先定义
MOVS  BUFFER2,ES:BUFFER1                 ;源操作数进行段重设
REP   MOVS   WORD PTR[DI],[SI]           ;用变址寄存器表示操作数
REP   MOVSB                              ;字节串传送
MOVSW                                    ;字串传送
```

但以下表示方法是非法的：

```
MOVSB  DEST,ES:SRC
```

串操作指令常常与重复前缀联合使用，这样不仅可以简化程序，而且提高了运行速度。但此时必须先把字符串的长度送CX寄存器中，以便控制指令结束。当然也可以单独使用串操作指令编制循环程序，不过此时需在程序中另加使CX寄存器减1的指令来控制循环次数。

例 4.8　将数据段中首地址为BUFFER1的200个字节传送到附加段首地址位BUFFER2的内存区中。使用字节串传送指令的程序段如下：

```
LEA   SI,BUFFER1          ;(SI)←源串首地址指针
LEA   DI,BUFFER2          ;(DI)←目的串首地址指针
MOV   CX,200              ;(CX)←字节串长度
CLD                       ;清方向标志DF
REP   MOVSB               ;传送200个字节
HLT                       ;停止
```

2. 串装入指令

指令格式：

```
LODS [seg:]src-string
LODSB
LODSW
```

LODS指令是将一个字符串中的字节或字逐个装入累加器AL或AX。指令的基本操作为：

① (AL)←((DS):(DI)) 或 (AX)←((DS):(DI))

② (DI)←(DI)±1（字节操作）

(DI)←(DI)±2（字操作）

其中，当方向标志DF=0时用+，当方向标志DF=1时用－。

LODS指令不影响状态标志位，而且一般不带重复前缀。因为将字符串的各个值重复地装入到累加器中没有什么意义。

例 4.9　内存中以BUFFER为首址的缓冲区内有10个非压缩型BCD码形式存放的十进制数，它们的值可能是0～9中的任意一个，将这些十进制数顺序显示在屏幕上。

在屏幕上显示一个字符的方法（详见本书第五章第八节的DOS系统功能调用部分）是：

```
MOV   AH,02H              ;(AH)←DOS系统功能号
MOV   DL,'Y'              ;(DL)←显示字符"Y"
INT   21H                 ;调用DOS的21H中断(在屏幕上显示)
```

根据题意可编程如下：

```
        LEA   SI,BUFFER           ;(SI)←缓冲区首址
```

```
        MOV   CX,10           ;(CX)←字符串长度
        CLD                   ;清状态标志位 DF
        MOV   AH,02H          ;(AH)←功能号
GET:    LODSB                 ;取一个 BCD 码到 AL
        OR    AL,30H          ;BCD 码转换为 ASCII 码
        MOV   DL,AL           ;(DL)←字符
        INT   21H             ;显示
        DEC   CX              ;(CX)←(CX)－1
        JNZ   GET             ;未完成 10 个字符则重复
        HLT
```

3. 串送存指令

指令格式：

[REP] STOS [ES:]dst—string

[REP] STOSB

[REP] STOSW

STOS 指令是将累加器 AL 或 AX 的值送存到内存缓冲区的某个位置上。指令的基本操作为：

① ((ES):(DI))←(AL) 或 ((ES):(DI))←(AX)

② (DI)←(DI)±1 (字节操作)

(DI)←(DI)±2 (字操作)

STOS 指令对状态标志位没有影响。指令若加上重复前缀 REP，则操作将一直重复进行下去，直到(CX)＝0。

例 4.10　将字符"＃"装入以 AREA 为首址的 100 个字节中。

```
LEA   DI,AREA
MOV   AX,'##'
MOV   CX,50
CLD
REP   STOSW
HLT
```

以上程序采用了送存 50 个字而不是送存 100 个字节的方法，虽然这两种方法程序执行的结果是相同的但以上程序的执行速度更快些。

例 4.11　一个数据块由大写或小写的英文字母、数字和各种其他符号组成，其结束符是回车符 CR(ASCII 码位 0DH)，数据块的首地址为 BLOCK1。将数据块传送到以 BLOCK2 为首地址的内存区，并将其中所用的英文小写字母(a～z)转换成相应的大写字母(A～Z)，其余不变。

前面已经讨论过英文小写字母与相应的大写字母的 ASCII 码之间有一定的关系，例如：

```
'a'＝61H        'A'＝41H
'b'＝62H        'B'＝42H
 ⋮               ⋮
'z'＝7AH        'Z'＝5AH
```

即只需将小写字母的ASCII码减20H，即可得到相应大写字母的ASCII码。程序见下：

```
        LEA   SI,BLOCK1        ;(SI)←源地址指针
        LEA   DI,BLOCK2        ;(DI)←目的地址指针
        CLD                    ;清方向标志DF
NEXT:   LODSB                  ;取一个字符到AL
        CMP   AL,0DH           ;是否回车符
        JZ    DONE             ;是，则转DONE
        CMP   AL,61H           ;否则，是否小于"a"
        JC    OK               ;是，则转OK
        CMP   AL,7BH           ;是否大于"z"
        JNC   OK               ;是，转OK
        SUB   AL,20H           ;否则，AL减20H
OK:     STOSB                  ;送存
        JMP   NEXT             ;转移到NEXT
DONE:   HLT                    ;停止
```

4. 串比较指令

指令格式：

[REPE/REPNE] CMPS [srg:]src-string,[ES:]dst-string

[REPE/REPNE] CMPSB

[REPE/REPNE] CMPSW

该指令是将两个字符串中相应的元素逐个进行比较(即相减)，但不将比较结果送回目的操作数，而反映在状态标志位上。CMPS指令对状态标志位SF、ZF、AF、PF、CF和OF有影响。指令的基本操作为：

① ((DS):(SI))－((ES):(DI))

② (SI)←(SI)±1，(DI)←(DI)±1 (字节操作)

(SI)←(SI)±2，(DI)←(DI)±2 (字操作)

CMPS指令与其他指令有所不同，指令中的源操作数在前，而目标操作数在后。另外，CMPS指令可以加重复前缀REPE(也可以写成REPZ)或REPNE(也可以写成REPNZ)，这是由于CMPS指令影响标志ZF。如果两个被比较的字节或字相等，则(ZF)＝1，否则(ZF)＝0。REPE或REPZ表示当(CX)≠0，且(ZF)＝1时继续进行比较。REPNE或REPNZ表示当(CX)≠0，且(ZF)＝0时继续进行比较。

如果想在两个字符串中寻找第一个不相等的字符，则应使用重复前缀REPE或REPZ，当遇到第一个不相等的字符时，就停止进行比较。但此时地址已被修改，即(DS:SI)和(ES:DI)已经指向下一个字节或字地址。应将SI和DI进行修正，使之指向所要寻找的不相等字符。同理，如果想要寻找两个字符串中第一个相等的字符，则应使用重复前缀REPNE或REPNZ。但是也有可能将整个字符串比较完毕仍未出现规定的条件(例如两个字符相等或不相等)，不过此时寄存器(CX)＝0，故可用条件转移指令JCXZ进行处理。

例4.12　比较两个字符串，找出其中第一个不相等字符的地址。如果两个字符全部相同，则转到ALLMATCH进行处理。这两个字符串长度均为20，首地址分别为STRING1和STRING2。

```
            LEASI,STRING1           ;(SI)←字符串 1 首地址
            LEA   DI,STRING2        ;(DI)←字符串 2 首地址
            MOV   CX,20             ;(CX)←字符串长度
            CLD                     ;清方向标志 DF
            REPE  CMPSB             ;如相等,重复进行比较
            JCXZ  ALLMATCH          ;若(CX)=0,跳至 ALLMATCH
            DEC   SI                ;否则(SI)-1
            DEC   DI                ;(DI)-1
            HLT                     ;停止
ALLMATCH:   MOV   SI,0
            MOV   DI,0
            HLT                     ;停止
```

5. 串扫描指令

指令格式:

[REPE/REPNE] SCAS [ES:]dst—string

[REPE/REPNE] SCASB

[REPE/REPNE] SCASW

SCAS 指令是在一个字符串中搜索特定的关键字。字符串的起始地址只能放在(ES:DI)中,不允许段超越。待搜索的关键字必须放在累加器 AL 或 AX 中。SCAS 指令的基本操作为:

① (AL)-((ES):(DI)) 或 (AX)-((ES):(DI))

② (DI)←(DI)±1 (字节操作)

(DI)←(DI)±2 (字操作)

SCAS 指令将累加器的内容与字符串中的元素逐个进行比较,比较结果也反映在状态标志位上。SCAS 指令将影响状态标志位 SF、ZF、AF、PF、CF 和 OF。如果累加器的内容与字符串中的元素相等,则比较之后(ZF)=1,因此,指令可以加上重复前缀 REPE 或 REPNE。前缀 REPE(即 REPZ)表示当(CX)≠0,且(ZF)=1 时继续进行扫描。而 REPNE(即 REPNZ)表示当(CX)≠0,且(ZF)=0 时继续进行扫描。

例 4.13　在包含 100 个字符的字符串中寻找第一个回车符 CR(其 ASCII 码为 0DH),找到后将其地址保留在(DS:DI)中,并在屏幕上显示字符“Y”。如果字符串中没有回车符,则在屏幕上显示字符“N”。该字符串的首地址位 STRING。根据要求可编程如下:

```
        LEA   DI,STRING         ;(DI)←字符串首址
        MOV   AL,0DH            ;(AL)←回车符
        MOV   CX,100            ;(CX)←字符串长度
        CLD                     ;清状态标志位 DF
        REPNE SCASB             ;如未找到,重复扫描
        JZ MATCH                ;如找到转 MATCH
        MOV DL,'N'              ;字符串中无回车,则(DL)←“N”
        JMP DSPY                ;转到 DSPY
MATCH:  DEC DI                  ;(DI)←(DI)-1
        MOV DL'Y'               ;(DL)←“Y”
```

```
DSPY:    MOV AH,02H
         INT 21H            ;显示字符
         HLT
```

在以上五条串操作指令中，有的指令有两个操作数，有的指令只有一个操作数。只有一个操作数时，有的指令是源操作数，有的指令是目标操作数。

关于串操作指令的重复前缀、操作数以及地址指针所用的寄存器等情况归纳如表 4-6 所示。

表 4-6 串操作指令的重复前缀、操作数和地址指针

指令	重复前缀	操作数	地址指针寄存器
MOVS	REP	目的，源	ES:DI,DS:SI
LODS	无	源	DS:SI
STOS	REP	目的	ES:DI
CMPS	REPE/REPNE	源，目的	DS:SI,ES:DI
SCAS	REPE/REPNE	目的	ES:DI

五、控制转移指令

8086/8088 指令系统提供了大量指令用于控制程序的流程。这类指令包括：转移指令、循环控制指令、过程调用指令和中断指令等四组。

下面分别进行讨论。

(一)转移指令

转移是一种将程序控制从一处改换到另一处的最直接的方法。在 CPU 内部，转移是通过将目的地址传送给 IP 来实现的。

转移指令包括无条件转移指令和条件转移指令。

1. 无条件转移指令

无条件转移指令的操作是无条件地将控制转移到指令中指定的目的地址。另外，目的地址可以用直接的方式给出，也可以用间接的方式给出。无条件转移指令对状态标志位没有影响。

(1)段内直接转移

指令格式及操作：

```
JMP  near-label ;(IP)←(IP)+disp(16 位)
```

指令的操作数是一个近标号，该标号在本段(或本组)内。指令汇编以后，计算出 JMP 指令的下一条指令的地址到目的地址之间的 16 位相对位移量 disp。指令的操作是将指令指针寄存器 IP 的内容加上相对位移量 disp，代码段寄存器 CS 的内容不变，从而使控制转移到目的地址。相对位移量可正可负，一般情况下，它的范围在－32768～＋32767 之间，故需用 2 个字节表示，加上一个字节的操作码，这种段内直接转移指令共有 3 个字节。请看下面几条指令。

```
      JMP  NEXT
      AND  AL,7FH
NEXT: XOR  AL,7FH
```

其中，NEXT 是本段内的一个标号，汇编语言计算出下一条指令(即 AND AL,7FH)的地址与标号 NEXT 代表的地址之间的相对位移量，执行 JMP NEXT 指令时，将上述位移量加到 IP 上，于是执行 JMP 指令之后接着就执行 XOR AL,7FH 指令，实现了程序的转移。

(2)段内直接短转移

指令格式及操作：

JMP Short—label ；(IP)←(IP)+disp(8 位)

段内直接短转移指令的操作数是一个短标号。此时，相对位移量 disp 的范围在 -128～+127 之间，只需用 1 个字节表示。段内直接短转移指令共有 2 个字节。

如果已知下一条指令到目的地址之间的相对位移量在 -128～+127 的范围内，则可在标号前写上运算符 SHORT，实现段内直接短转移。但是，对于一个段内直接转移指令，如果相对位移量的范围在 -128～+127 之间，而且目标地址的标号已经定义(即标号先定义后引用，这种情况称为向后引用的符号)，那么即使在标号前没有写上运算符 SHORT，汇编程序也能自动生成一个 2 字节的短转移指令。这种情况属于隐含的短转移。如果向前引用标号(即标号先引用，后定义)，则标号前应写上运算符 SHORT，否则，即使位移量的范围不超过 -128～+127，汇编后仍会生成一个 3 字节的近转移指令。例如：

* 向后引用的标号，可不写运算符 SHORT

```
target:  …
         :                      ;先定义标号 target
                                ;相对位移量不超过-128～+127
         JMP  target            ;后引用标号 target
```

* 向前引用的标号，应写明 SHORT

```
         JMP SHORT target       ;先引用标号 target
         :                      ;相对位移量不超过-128～+127
target:  …                      ;此处定义标号 target
```

(3)段内间接转移

指令格式及操作：

JMP reg16/mem16 ；(IP)←(reg16)/(IP)←(mem16)

指令的操作是一个 16 位的寄存器(reg16)或存储器(mem16)地址。存储器可用各种寻址方式。指令的操作是用指定的寄存器或存储器中的内容作为目标的偏移地址取代原来 IP 的内容，以实现程序的转移。由于是段内转移，故 CS 寄存器的内容不变。下面是几条段内间接转移指令。

```
JMP  AX
JMP  SI
JMP  TABEL[BX]
JMP  ALPHA—WORD
JMP  WORD PTR[BP][DI]
```

上面前两条指令的操作数是 16 位寄存器。第 3、4 条指令的操作数应是已被定义成 16 位的存储器。第 5 条指令利用运算符"PTR"将存储器操作数定义为 WORD(字，即 16 位)、关于"PTR"的说明请参阅本书第五章第二节第二小节中表达式部分关于分析运算符和合成运算符的介绍。

(4)段间直接转移

指令格式及操作：

JMP far-label ；(IP)←OFFSET far-label

;(CS)←SEG far-label

指令的操作数是一个远标号,该标号在另一个代码段内。指令的操作是将标号的偏移地址取代指令指针寄存器 IP 的内容,同时将标号的段基值取代段寄存器 CS 的内容,结果使控制转移到另一代码段内指定的标号处。例如:

JMP LABEL—DECLARED—FAR

JMP FAR PTR LABEL—NAME

上面第一条指令中的 LABEL—DECLARED—FAR 应是一个在另一代码段内已定义的远标号。第二条指令利用运算符 PTR 将标号 LABEL—NAME 的属性指定为 FAR。

(5)段间间接转移

指令格式及操作:

JMP mem32 ;(IP)←(mem32)

;(CS)←(mem32+2)

指令的操作数是一个 32 位的存储器(mem32)地址,指令的操作是将存储器的前两个字节送到 IP 寄存器,存储器的后两个字节送到 CS 寄存器,以实现到另一个代码段的转移。

需注意一点,段间的间接转移指令的操作数不能是寄存器。

以下是段间间接转移指令的例子,

JMP VAR—DOUBLEWORD

JMP DWORD PTR[BP][DI]

上面第一条指令中,VAR—DOUBLEWORD 应是一个已经定义成 32 位的存储器变量(例如可用数据定义伪指令 DD 定义)。第二条指令中,利用运算符 PTR 将存储器操作数的类型定义成 DWORD(双字,即 32 位)

2. 条件转移指令

指令格式为:

JCC short-babel

在汇编语言程序设计中,常利用条件转移指令来构成分支程序。指令助记符中的"CC"表示条件。这种指令的执行包括两个过程:第一步,测试规定的条件;第二步,如果条件满足,则转移到目的地址;否则,继续顺序执行。

条件转移指令也只有一个操作数,用以指明转移的目的地址。但是它与无条件转移指令 JMP 不同,条件转移指令的操作数必须是一个短标号,也就是说,所有的条件转移指令都是 2 字节指令,转移指令的下一条指令到目标地址之间的距离必须在－128～＋127 的范围内。如果指令规定的条件满足,则将这个位移量加到 IP 寄存器上,即(IP)←(IP)＋disp,实现程序的转移。

绝大多数条件转移指令(除 JCXZ 指令外)将状态标志位的状态作为测试的条件。因此,首先应执行影响有关的状态标志位的指令,然后才能用条件转移指令测试这些标志,以确定程序是否转移。CMP 和 TEST 指令常常与条件转移指令配合使用,因为这两条指令不改变目标操作数的内容,但可以影响状态标志位。其他如加法、减法及逻辑运算指令等等也影响状态标志位。

8086/8088 的条件转移指令非常丰富,不仅可以测试一个状态标志位的状态,而且可以综合测试几个状态标志位;不仅可以测试无符号数的高低,而且可以测试带符号数的大小,等等,编程时使用十分灵活、方便。下面,将所有的条件转移指令的名称、助记符及转移条件列于表

4 7 中。

表 4-7 条件转移指令

指 令 名 称	助记符	转 移 条 件	备 注
等于/零转移	JE/JZ	(ZF)=1	
不等于/非零转移	JNE/JNZ	(ZF)=0	
负转移	JS	(SF)=1	
正转移	JNS	(SF)=0	
偶转移	JP/JPE	(PF)=1	
奇转移	JNP/JPO	(PF)=0	
溢出转移	JO	(OF)=1	
不溢出转移	JNO	(OF)=0	
进位转移	JC	(CF)=1	
不进位转移	JNC	(CF)=0	
低于/不高于或等于转移	JB/JNAE	(CF)=1	无符号数
高于或等于/不低于转移	JAE/JNB	(CF)=0	无符号数
高于/不低于或等于转移	JA/JNBE	(CF)=0 且(ZF)=0	无符号数
低于或等于/不高于转移	JBE/JNA	(CF)=1 或(ZF)=1	无符号数
大于/不小于或等于转移	JG/JNLE	(SF)=(OF)且(ZF)=0	带符号数
大于或等于/不小于转移	JGE/JNL	(SF)=(OF)	带符号数
小于/不大于或等于转移	JL/JNGE	(SF)≠(OF)且(ZF)=0	带符号数
小于或等于/不大于转移	JLE/JNG	(SF)≠(OF)或(ZF)=1	带符号数
CX 等于零转移	JCXZ	(CX)=0	

表 4-7 中同一行内用斜杠隔开的几个助记符，实质上代表同一条指令的几种不同的表示方法。

下面再来看几个应用条件转移指令的程序例子

例 4.14 在内存的数据段中存放了若干个 8 位带符号数，数据块的长度为 COUNT(不超过 255)，首地址为 TABLE，试统计其中正元素、负元素及零元素的个数，并分别将个数存入 PLUS、MINUS 和 ZERO 单元。

为了统计正元素、负元素和零元素的个数，可先将 PLUS、MINUS 和 ZERO 三个单元清零，然后将数据表中的带符号数逐个取入 AL 寄存器并使其影响状态标志位，再利用前面介绍的 JS、JZ 等条件转移指令测试该数是一个负数、零还是正数，然后分别在相应的单元中进行计数。编程如下：

```
XOR   AL,AL          ;(AL)←0
MOV   PLUS,AL        ;清 PLUS 单元
MOV   MINUS,AL       ;清 MINUS 单元
MOV   ZERO,AL        ;清 ZERO 单元
LEA   SI,TABLE       ;(SI)←数据表首址
MOV   CX,COUNT       ;(CX)←数据表长度
```

```
            CLD                 ;清状态标志位 DF
CHECK:      LODSB               ;取一个数据到 AL
            OR    AL,AL         ;使数据影响状态标志位
            JS    X1            ;如为负,转 X1
            JZ    X2            ;如为零,转 X2
            INC   PLUS          ;否则为正,PLUS 单元加 1
            JMP   NEXT
X1:         INC   MINUS         ;MINUS 单元加 1
            JMP   NEXT
X2:         INC   ZERO          ;ZERO 单元加 1
NEXT:       LOOP  CHECK         ;CX 减 1,如不为零,则转 CHECK
            HLT                 ;停止
```

以上程序中的 LOOP CHECK 指令是一条循环控制指令,它的操作是将 CX 寄存器的内容减 1,如结果不等于零,则转移到短标号 CHECK。后面很快就要讨论这条指令。

例 4.15　在以 DATA 为首址的内存数据段中,存放了 100 个 16 位带符号数,试将其中最大和最小的带符号数找出来,分别存放到以 MAX 和 MIN 为首的内存单元中。

为了寻找最大和最小的元素,可先取出数据块中的一个数据作为标准,暂且将它同时存放到 MAX 和 MIN 单元中,然后将数据块中的其它数据逐个分别与 MAX 和 MIN 中的数相比较,凡大于 MAX 者,取代原来 MAX 中的内容,凡小于 MIN 者,取代原来 MIN 中的内容,最后即可得到数据块中最大和最小的带符号数。

必须注意,比较带符号数的大小时,应该采用 JG 和 JL 等条件转移指令,根据要求可编程如下:

```
            LEA   SI,DATA       ;(SI)←数据块首址
            MOV   CX,100        ;(CX)←数据块长度
            CLD                 ;清方向标志 DF
            LODSW               ;取一个 16 位带符号数到 AX
            MOV   MAX,AX        ;送 MAX 单元
            MOV   MIN,AX        ;送 MIN 单元
            DEC   CX            ;(CX)-1
NEXT:       LODSW               ;取下一个 16 位带符号数
            CMP   AX,MAX        ;与 MAX 单元内容比较
            JG    GREATER       ;大于 MAX,则转 GREATER
            CMP   AX,MIN        ;否则,与 MIN 单元内容比较
            JL    LESS          ;小于 MIN,则转 LESS
            JMP   GOON          ;否则,转 GOON
GERATER:    MOV   MAX,AX        ;(MAX)←(AX)
            JMP   GOON          ;转 GOON
LESS:       MOV   MIN,AX        ;(MIN)←(AX)
GOON:       LOOP  NEXT          ;CX 减 1,若不等于零,转 NEXT
            HLT                 ;停止
```

（二）循环控制指令

8086/8088 指令系统专门设计了几条循环控制指令，用于使一些程序段反复执行，形成循环程序。循环控制指令有以下几条：

1. LOOP

指令格式：

LOOP short-label

LOOP 指令规定将 CX 寄存器作为计数器，指令的操作是先将 CX 的内容减 1，如结果不等于零，则转到指令中指定的短标号处；否则，顺序执行下一条指令。因此，在循环程序开始前，应将循环次数送 CX 寄存器。指令的操作只能是一个短标号，即跳转距离不超过 −128～+127 的范围。LOOP 指令对状态标志位没有影响。LOOP 指令的应用可参阅前面的例 4.14 和例 4.15。

2. LOOPE/LOOPZ

指令格式：

LOOPE short-label/LOOPZ short-label

以上两种格式实际上代表同一条指令。本指令的操作也是先将 CX 寄存器的内容减 1，如结果不为零，且零标志(ZF)＝1，则转移到指定的短标号。LOOPE/LOOPZ 指令对状态标志位也没有影响。

这条指令是有条件地形成循环，即当规定的循环次数尚未完成时，还必须满足“相等”或者“等于零”的条件，才能继续循环。

例：4-16　比较两组输入端口的数据是否一致，其中一组端口的首地址为 MAIN—PORT；另一组端口的首地址为 REDUNDANT—PORT。两组端口的数目均为 NUMBER。

```
        MOV   DX,MAIN—PORT            ;(DX)←主端口地址指针
        MOV   BX,REDUNDANT—PORT       ;(BX)←冗余端口地址指针
        MOV   CXM,NUMBER              ;(CX)←端口数
TOP:    IN   AX,DX                    ;输入主端口数据到 AX
        XCHG   AX,BP                  ;主端口数据暂存 BP
        INC   DX                      ;主端口地址指针加 1
        XCHG   BX,DX                  ;BX,DX 交换
        IN   AX,DX                    ;输入其余端口数据到 AX
        INC   DX                      ;冗余端口地址指针加 1
        XCHG   BX,DX                  ;恢复 BX,DX
        CMP   AX,BP                   ;比较两个端口的数据
        LOOPE   TOP                   ;如两端口数据相等且(CX)≠0,则转 TOP
        JNZ PORT—DISPUTE              ;如两端口数据不等,转 PORT—DISPUTE
          ⋮
PORT—DISPUTE:
```

3. LOOPNE/LOOPNZ

指令格式：

LOOPNE short-label/LOOPNZ short-label

本指令同样也有两种表示形式。指令的操作是将 CX 寄存器的内容减 1，如结果不为零，且

零标志(ZF)＝0(表示"不相等"或"不等于零")，则转移到指定的短标号。这条指令对状态标志位也没有影响。

(三)过程调用指令

如果有一些程序段需要在不同的地方多次反复地出现，则可以将这些程序段设计成为过程(相当于子程序)，每次需要时进行调用。过程结束后，再返回原来调用的地方。采用这种方法不仅可以使源程序的总长度大大缩短，而且有利于实现模块化的程序设计，使程序的编制、阅读和修改都比较方便。

被调用的过程可以在本段内(近过程)，也可在其他段(远过程)。调用的过程地址可以用直接的方式给出，也可用间接的方式给出。过程调用指令和返回指令对状态标志位都没有影响。

1. 过程调用指令

(1)段内直接调用

指令格式及操作：

```
CALL near-proc          ;(SP)←(SP)-2,((SP)+1:(SP))←(IP)
                        ;(IP)←(IP)+disp
```

指令的操作数是一个近过程，该过程在本段内。指令汇编以后，得到CALL的下一条指令与被调用的过程入口地址之间的16位相对位移量disp。指令的操作是将指令指针IP压入堆栈，然后将相对位移量disp加到IP上，使控制转到调用的过程。相对位移量的范围为－32768～＋32767，占2个字节，段内直接调用指令共有3个字节。

(2) 段内间接调用

指令格式及操作：

```
CALL  reg16/mem16       ;(SP)←(SP)-2, ((SP)+1:(SP))←(IP)
                        ;(IP)←(reg16)/(mem16)
```

指令的操作数是一个16位的寄存器或存储器，其中的内容是一个近过程的入口地址。本指令将IP寄存器压入堆栈，然后将寄存器或存储器的内容传送到IP。

(3)段间直接调用

指令格式及操作：

```
CALL far-proc           ;(SP)←(SP)-2, ((SP)+1:(SP))←(CS)
                        ;(CS)←SEG far-proc
                        ;(SP)←(SP)-2, ((SP)+1:(SP))←(IP)
                        ;(IP)←OFFSET far-proc
```

指令的操作数是一个远过程，该过程在另外的代码段内。段间直接调用指令先将CS中的段基值压入堆栈，并将远过程所在的段值SEG far-proc送CS；再将IP中的偏移地址压入堆栈，然后将远过程的偏移地址OFFSET far-proc送IP。

(4)段间间接调用

指令格式及操作：

```
CALL  mem32     ;(SP)←(SP)-2, ((SP)+1:(SP))←(CS)
                ;(CS)←(mem32+2)
                ;(SP)←(SP)-2, ((SP)+1:(SP))←(IP)
                ;(IP)←(mem32)
```

指令的操作数是一个32位的存储器地址，指令的操作是先将CS寄存器压入堆栈，并将

存储器的后两个字节送 CS；再将 IP 寄存器压入堆栈，然后将存储器的前两个字节送 IP，于是控制转到另一个代码段的远过程。

2. 过程返回指令

指令格式及操作：

(1)从近过程返回

RET　　　　　　　;(IP)←((SP)+1:(SP))，(SP)←(SP)+2

RET pop-value　　;(IP)←((SP)+1:(SP))，(SP)←(SP)+2

　　　　　　　　　;(SP)←(SP)+pop-value

(2)从远过程返回

RET　　　　　　　;(IP)←((SP)+1:(SP))，(SP)←(SP)+2

　　　　　　　　　;(CS)←((SP)+1:(SP))，(SP)←(SP)+2

RET pop-value　　;(IP)←((SP)+1:(SP))，(SP)←(SP)+2

　　　　　　　　　;(CS)←((SP)+1:(SP))，(SP)←(SP)+2

　　　　　　　　　;(SP)←(SP)+pop-value

过程体中一般总是包含返回指令 RET，它将堆栈中的断点弹出，控制程序返回到原来调用过程的地方。通常，REP 指令的类型是隐含的，它自动与过程定义时的类型匹配，如为近过程，返回时将栈顶的字弹出到 IP 寄存器；如为远过程，返回时先从栈顶弹出一个字到 IP，接着再弹出一个字到 CS。但是，当采用间接调用时，必须注意保证 CALL 指令的类型与过程中 RET 指令的类型匹配，以免发生错误。例如：CALL　WORD PTR [BX]只能调用一个近短程，而 CALL DWORD PTR [BX]能够调用一个远过程。

此外，RET 指令还允许带一个弹出值(pop-value)，这是一个范围为 0～64K 的立即数，通常是偶数。弹出值表示返回时从堆栈中舍弃的字节数。例如：RET 4，返回时舍弃堆栈中的 4 个字节。这些字节一般是调用前通过堆栈向过程传递的参数。

六、处理器控制指令

这一类指令用于对 CPU 进行控制，例如对 CPU 中某些状态标志位的状态进行操作，以及使 CPU 暂停、等待等等。8086/8088 指令系统的处理器控制指令可分为三组。

(一)标志位操作指令

1. CLC

清进位标志。指令的操作为(CF)←0。

2. STC

置进位标志。指令的操作为(CF)←1。

3. CMC

对进位标志求反。指令的操作为(CF)←(CF)。

4. CLD

清方向标志。指令的操作为(DF)←0。

5. STD

置方向标志。指令的操作为(DF)←1。

6. CLI

清中断允许标志。指令的操作为(IF)←0。

7. STI

置中断允许标志。指令的操作为(IF)←1。在执行这条指令后,CPU 将允许外部的可屏蔽中断请求。

这些指令仅对有关状态标志位执行操作,而对其他状态标志位则没有影响。

(二)外部同步指令

1. HLT

执行 HLT 指令后,CPU 进入暂停状态。外部中断(当(IF)=1 时的可屏蔽中断请求 INTR,或非屏蔽中断请求 NMI)或复位信号 RESET 可使 CPU 退出暂停状态。HLT 指令对状态标志位没有影响。

2. WAIT

如果 8086/8088CPU 的$\overline{\text{TEST}}$引脚上的信号无效(即高电平),则 WAIT 指令使 CPU 进入等待状态。一个被允许的外部中断或$\overline{\text{TEST}}$信号有效,可使 CPU 退出等待状态。

在允许中断的情况下,一个外部中断请求将使 CPU 离开等待状态,转向中断服务程序。此时被推入堆栈进行保护的断点地址即是 WAIT 指令的地址,因此从中断返回后又执行 WAIT 指令,CPU 再次进入等待状态。

如果$\overline{\text{TEST}}$信号变低(有效),则 CPU 不再处于等待状态,开始执行下面的指令。但是,在执行完下一条指令之前,不允许有外部中断。

本指令对状态标志位没有影响。WAIT 指令的用途是使 CPU 本身与外部的硬件同步工作。

3. ESC

指令格式:

```
ESC   ext-op,src
```

ESC 指令使其他处理器可使用 8086/8088 的寻址方式,并从 8086/8088CPU 的指令队列中取得指令。以上指令格式中的 ext-op 是其他处理器的一个操作码(外操作码),src 是一个存储器操作数。执行 ESC 指令时,8086/8088CPU 访问一个存储器操作数,并将其放在数据总线上,供其他处理器使用。此外没有其他操作。例如协处理器 8087 的所有指令机器码的高五位都是"11011",而 8086/8088 的 ESC 指令机器码的第一个字节恰是"11011XXX",因此,对于这样的指令,8086/8088CPU 将其视为 ESC 指令,它将存储器操作数置于总线上,然后由 8087 来执行该指令,并使用总线上的操作数。8087 的指令系统请参考有关资料。

ESC 指令对状态标志位没有影响。

4. LOCK

这是一个特殊的可以放在任何指令前面的单字节前缀。这个指令前缀迫使 8086/8088CPU 的总线锁定信号线$\overline{\text{LOCK}}$维持低电平(有效),直到执行完下一条指令。外部硬件可接收这个$\overline{\text{LOCK}}$信号。在其有效期间,禁止其他处理器对总线进行访问。共享资源的多处理器系统中,必须提供一些手段对这些资源的存取进行控制,指令前缀 LOCK 就是一种手段。

(三)空操作指令 NOP

执行 NOP 指令时不进行任何操作,但占用 3 个时钟周期,然后继续执行下一条指令。NOP 指令对状态标志位没有影响,指令没有操作数。

第四节　80X86扩充的和增加的指令

上一节，我们对 8086/8088 的指令系统作了详尽的阐述，以下将介绍标准 16 位 CPU80286、32 位 CPU80386 和 80486 的指令系统。由于 80286CPU 对 8086/ 8088 的指令是向上兼容的，即 80286 的指令系统是 8086/8088 指令的高级集，而 80386、80486 是 80286 的增强型，它们也是对 8086/8088 的指令系统向上兼容，因此本节仅介绍 80286、80386 和 80486 的新增指令以及在 8086/8088 基础上扩充了新的功能的一些指令。除特别说明为保护模式下的指令外，缺省则为实地址模式下的指令。

一、80286扩充的和增加的指令

80286 指令系统除了包含 8086/8088 的全部指令外，新增指令以及增强了功能的指令见表 4-8。其中控制保护态指令是 80286 工作在保护模式下的一些特权方式指令，常用于操作系统及其它的控制软件中，应用程序设计中用到的不多，故不作详细介绍。其余指令功能介绍如下。

表 4-8　80286 增强与增加的指令

类　别	扩　充	增　加
数据传送指令	PUSH data	PUSHA/POPA
算术运算指令	IMUL reg16,data IMUL reg16,reg16/mem16,data	
逻辑运算与移位指令	SHL dst,count(1～31) 其余如 SAL,SAR,SHR,ROL,ROR,RCL,RCR 七条指令同 SHL.	
串操作指令		[REP] INS/[REP] OUTS
高级语言指令		BOUND reg16,mem16 ENTER data16,data8 LEAVE
控制保护指令	LAR (设置访问权限) LGDT(设置全局描述符表) LIDT(设置中断描述符表) LLDT(设置局部描述符表) LTR (设置任务寄存器) LMSW(设置机器状态字) VERR(带校验读) ARPL(调整已请求特权级)	LSL (设置段限值) SGDT(保存全局描述符表) SIDT(保存中断描述符表) SLD (保存局部描述符表) DTR (保存任务寄存器) DMDW(保存机器状态字) VERW(带校验写) CLTS(清除任务转移标志)

(一)堆栈指令(PUSHA/POPA 及 PUSH data)

1. PUSHA/POPA 指令将所有通用寄存器的值压入/弹出堆栈，压入的顺序是:AX、CX、DX、BX、SP、BP、SI、DI(SP 是执行该指令之前的值)，弹出的顺序与压入的相反。将这对指令用于子程序调用(或中断响应)时的现场保护与恢复，比用 PUSH 和 POP 分别压入和弹出寄存

器要快的多。

2. PUSH data 指令是将数字 data(立即数)压入堆栈,该数字的范围在 0～65535 或 －32768～＋32767,如果给出的数不够 16 位,它会在自动扩展后压入堆栈。

(二)乘法指令(IMUL)

在 80286 中允许该指令有两个或三个操作数。

1. IMUL reg16,data ;(reg16)←(reg16)×data

2. IMUL reg16,reg16/mem16,data ;(reg16)←(reg16/mem16)×data

式中立即数的范围是－32768～＋32767 的整数,并限制乘后结果是 16 位有符号数,如果溢出,则溢出部分丢掉,并置 CF＝OF＝1。

例:IMUL AX,10;

IMUL DI,[BX+TABLE],3;

IMUL BX,CX,5

(三)移位指令(SHL dst,count)等

在 8086/8088 中规定所有 8 条移位和循环移位指令中计数值部分或是常数 1 或是 CL 中内容所规定的次数。80286 扩充计数值可为常数,但其范围在 1～31 之间。

例:SAR BX,14; ROR AL,4

(四)串输入/输出指令(INS/OUTS)

1. INS/OUTS 指令的形式有:

(1) [REP] INS [ES:]det-string,DX/OUTS DX,[sreg:]sre-string

(2)[REP] INSB/OUTSB

(3) [REP] INSW/OUTSW

其中,(3)要求有操作数部分,指令功能同 IN/OUT。(2)(3)分别实现 DX 指定的端口与由指定的内存地址之间的数据块传送,其类型可以是字节或字。如果加重复前缀 REP,完成整个串的输入/输出操作,这时 CX 寄存器中为重复前缀操作的次数。

(五)高级语言类指令

此类中的三条指令类似于高级语言中的指令。

1. 数组边界检查指令 (BOUND reg16,mem16)

BOUND 指令验证在指定寄存器中的第一个操作数是否在第二个操作数(存储器中)所指向的两个界限内,若不在,则产生一个5号中断。指令中假定上、下界值(即数组的起始和结束地址)依次存放在相邻存储单元中。

2. 进入和退出过程指令

在许多高级语言中,每个子程序(或函数)都有自己的局部变量。当这些子程序在执行时,这些局部变量才有意义。为保存这些局部变量,当执行到这些子程序时,应为其局部变量建立起相应的堆栈框架,80286用 ENTER 指令来完成此功能。在退出子程序时,应撤除这个框架,80286用 LEAVE 指令来完成。

(1)ENTER data16,data8

该指令为过程参数建立一个堆栈区,指令中第一操作数指出过程所要使用的堆栈字节数,第二操作数指出过程的嵌套层数:0～31。

(2)LEAVE 指令的功能是释放上述 ENTER 指令所设置的堆栈空间。

```
TASK    PROC  NEAR
```

```
        ENTER   200,8        ;建立堆栈空间为200字节,允许过程嵌套8层
        ⋮
        LEAVE                ;释放堆栈空间
        RET
TASK    ENDP
```

二、80386、80486扩充的和增加的指令

80386对8086/8088和80286指令系统进行了扩充。这种扩充不仅体现在增加了指令的种类、增强了一些指令的功能,也体现在提供了32位寻址方式和32位操作方式。80486是80386体系结构的基础上进行了一些扩展,增加了一些相应的指令。所以,80486指令系统是 8086/8088、80286、80386指令系统的最高集,所有从8086/8088、80286延伸而来的指令均使用于80386、80486的32位寻址方式和32位操作方式。表4-9示出了80386、80486 扩充与增加的指令,由于篇幅所限,下面仅介绍新增指令的功能。

表4-9　80386、80486扩充与增加的指令

类　别	80386		80486
	扩　充	增　加	增　加
数据传送指令	PUSHAD/POPAD PUSHFD/POPFD		BSWAP reg32 CMPXCHG reg/mem,reg
算术运算指令	IMUL reg,reg/mem CWDE CDQ		XADD reg/mem,reg
逻辑运算与移位指令	所有串操作指令 后面扩展 D,如: MOVSD,OUTD…		
位运算指令		BT/BTC/BTR/BTS dst,reg/count BSF/BSR reg,reg/mem	
条件设置指令		SET　if reg/mem	
Cache 管理			INVD WBINVD INVLPG

(一)数据传送指令

1. 数的传送与扩展 (MOVSX/MOVZX)

MOVSX(MOVZX)传送有(无)符号数到目标寄存器去,并将符号(0)扩展到操作数的所有位去。

2. 字节交换指令 (BSWAP)

用于将32位通用寄存器中的双字以字节为单位高、低字节进行交换。

3. 比较与交换指令 (CMPXCHG)

CMPXCHG 将存放在8位、16位或32位寄存器或存储器中的第一操作数与累加器 AL、AX 或 EAX 的内容进行比较；如果相等，则 ZF＝1，并将第二操作数送第一操作数单元；否则，ZF＝0，并将第一操作数送相应的累加器。

（二）算术运算指令

XADD 是交换与相加指令，它将第一操作数（存放在8位、16位或32 位寄存器或存储单元中）与第二操作数（寄存器中）相加，结果取代第一操作数，并将第一操作数取代第二操作数。

（三）逻辑运算与移位指令

SHRD/SHLD 是双精度右（左）移指令，它将指定的一些位右（左）移到一个操作数中。这两条指令有三个操作数，第一个操作数是一个16位或32位的数（存于寄存器或内存中），它将接收移动的位，第二个操作数是包含有要移动位的操作数，第三个操作数是要移动的位数（在 CL 中，或是立即数）。例如：

```
MOV AX,3AF2H
MOV BX,9C00H
SHLD AX,BX,7 ;AX 左移7位,BX 左移7位,并移入 AX 中空出的7位中,
                即 AX=794EH
```

（四）位操作指令

1. 测试与置位指令（BT/BTC/BTR/BTS）

BT 指令用于检查指定位，并将该位复制到进位标志中。

```
例：  MOV   CX,4
      BT    [BX],CX ;检查由 BX 指向的数的位4,且位4放入进位标志中
```

BTC 指令用于检查指定位，将其取反，并复制到进位标志中。

BTR/BTS 用于检查指定位，并将该位复制到进位标志中，然后将指定位清0（置1）。

2. 位扫描指令（BSF/BSR）

BSF 用于对第二操作数从低位到高位进行扫描，并将扫描到的第一个“1”的位号送入目标寄存器。如果位串中所有的位均为0，则将 ZF 标志位置1，否则将 ZF 清0。

BSR 指令的功能同 BSF，只是从高位到低位进行反向扫描。

（五）条件设置指令（SET）

SET 指令用于测试指定的状态标志位所处的状态，并根据测试的结果，将指定的第二操作数置1或置0。此类指令有：SETZ；SETNZ 等等。

（六）Cache 管理指令

这类指令用于80486管理 CPU 内部的8K Cache。

1. 作废 Cache 指令（INVD）

用于将 Cache 的内容作废。其具体操作是：刷新内部 Cache，并分配一个专用总线周期刷新外部 Cache。执行该指令不会将外部 Cache 中的数据写回主存。

2. 写回和作废 Cache 指令（WBINVD）

WBINVD 先刷新内部 Cache，并分配一个专用总线周期将外部 Cache 的内容写回主存，并在此后的一个总线周期将外部 Cache 刷新。

3. 作废 TLB 项指令（INVLPG）

该指令用于使 TLB 中的某一项作废。如果 TLB 中含有一个存储器操作数映像的有效项，则该 TLB 项被标记为无效。

思考题与习题

1. 设 BX＝637DH,SI＝2A9BH,位移量 D＝7237H,试求下列寻址下有效地址 EA＝?。

   (1)立即寻址　　(2) 直接寻址

   (3)基址加变址寻址　　(4) 用 BX 间接寻址

2. 假定 LAB 是标号,VAR 是变量,CON 是常数名称,列出下述每个操作数所有可能的寻址方式。

   (1)VAR[BX]　　(2) CON＋50H　　(3) VAR

   (4)LAB　　(5) VAR[BX＋3]　　(6) VAR[BX][DI]

3. 试指出下列传送类指令的寻址方式。(所有标识符都是字变量)

   (1)MOV　DI,1000　　(2) MOV　VAR[BX],CX

   (3)MOV　AX,VAR[BX][DI]　　(4) MOV　[BX＋100],DI

   (5)MOV [BP][SI],100　　(6) PUSH AX

4. 指出下列传送指令中非法的指令。

   (1)MOV　BX,AL　　(2) MOV　BH,AL

   (3)MOV　100,CL　　(4) MOV　CL 100

   (5)MOV　SS,2400H　　(6) MOV　CS,2400H

   (7)XCHG　AH,AL　　(8) XCHG　200,AL

   (9)OUT　21H,AL　　(10) OUT　260H,AL

5. 若(SP)＝2000H,(SX)＝3355H,(BX)＝4466H，试指出下列指令或程序段执行后有关寄存器的内容。

   (1)PUSH　AX 执行后(AX)＝?,(SP)＝?

   (2)PUSH　AX
      PUSH　BX
      POP　DX

执行后,(AX)＝?,(DX)＝?,(SP)＝?

6. 设(BX)＝6F30H,(BP)＝0200H,(SI)＝0046H,(SS)＝2F00H,[2F246H]＝ 4154H，试求执行 XCHG　BX,[BP+DI]后,(BX)＝?,[2F246H]＝?

7. 设(BX)＝0400H,(SI)＝003CH,执行 LEA BX,[BX＋DI—0F62H]后,(BX)＝?

8. 设 DS＝C000H,[C0010H]＝0180H,"[C0012H]＝2000H,问执行 LDS　SI,[10H]后,(SI)＝?,(DS)＝?

9. 已知(DS)＝091DH,(SS)＝1E4AH,(AX)＝1234H,(BX)＝0024H,(CX)＝5678H,(BP)＝0024H,(SI)＝0012H,(DI)＝0032H,[09226]＝00F6H,[09228]＝1E40H,[1EAF6]＝091DH，试求单独执行下列指令后的结果?

```
(1)MOV   CL,20H[BX][SI]        ;(CL)＝?
(2) MOV  [BP][DI],CX           ;[1E4F6H]＝?
(3)LEA   BX,20H[BX][SI]        ;(BX)＝?
   MOV   AX,2[BX]              ;(AX)＝?
(4)LDS   SI,[BX][DI]           ;
   MOV   [SI],BX               ;[SI]＝?
```

(5)XCHG CX,32H[BX] ;

XCHG 20[BX][SI],AX ;(AX)=? [09226H]=?

10. 使用移指令来做乘以2和除以2是很方便的。试把+53、−49乘以2，它们各应用什么指令，得到的结果各是什么?若除以2呢?

11. 若 CPU 中各寄存器及 RAM 参数如图4-22所示，试求独立执行如下指令后，CPU 及 RAM 相应寄存器及存储单元的内容为多少?

	CPU	CPU	
CS	3000H	FFFFH	CX
DS	2050H	0004H	BX
SS	50A0H	00000	SP
ES	0FFFH	17C6H	DX
IP	0000H	8094H	AX
DI	000AH	1403H	BP
SI	0008H	1	CF

RAM	执行前	执行后
20506H	06	
20507	00	
20508	87	
20509	15	
2050A	37	
2050B	C5	
2050C	2F	

图4-22

(1)MOV DX,[BX]2 ; DX= BX=

(2) PUSH CX ; SP= [SP]=

(3)MOV CX,BX ; CX= BX=

(4)TEST AX,01 ; AX= CF=

(5)MOV AL,[SI] ; AL=

(6)ADC AL,[DI] ; AL= CF=

DAA ; AL=

(7)INC SI ; SI=

(8)DEC DI ; DI=

(9)MOV [DI],AL ; [DI]=

(10) XCHG AX,DX ; AX= DX=

(11) XOR AH,BL ; AH= BL=

(12) JMP DX ; IP=

12. 设 DS=2000H，BX=1256H，SI=528FH，偏移量=20AH，[232F7H]=3280H，[264E5H]=2450H，执行下述指令

(1)JMP BX ; IP=?

(2)JMP TABLE[BX] ; IP=?

(3)JMP [BX][SI] ; IP=?

13. 8086/8088用什么途径来更新 CS 和 IP 的值?

14. 设(IP)=3D8FH，(CS)=4050H，(SP)=0F17CH，当执行 CALL 2000:0094H 后，试指出(IP)、(CS)、(SP)、[SP]、[SP+1][SP+2]和[SP+3]的内容?

15. 已知一个关于0～9的数字的 ASCII 码表首址是当前数据段的0A80H，现要找出数字5的 ASCII 码，试写出用指令 XCHG 进行翻译的指令序列。

16. 有程序段

```
MOV   CX,10
LEA   SI,First
LEA   DI,Second
REP   MOVSB
```

请指出该程序段的功能。

17. 今有以下程序段

```
CLD
LEA   DI,[0404H]
MOV   DS,0080H
XOR   AX,AX
REP   STOSW
```

试分析执行该程序段的功能。

18. 试编程完成(AX)×5/2的程序段。

19. 若(AL)=FFH,(BL)=03H,指出下列指令程序后标志 OF、SF、ZF、PF、CF 的状态。

(1)ADD BL,AL　　(2) INC BL

(3)SUB BL,AL　　(4)NEG BL

(5)CMP BL,AL　　(6) MUL BL

(7)AND BL,AL　　(8)IMUL BL

(9)OR BL,AL　　(10) SHL BL,1

(11)XOR BL,BL　　(12) SAR AL,1

(13)SHR AL,1

20. 试编写一程序段,根据 AL 中的内容决定程序的走向。若位0等于1,其它位为0,转向 LAB1;若位1是1,其它位为0,则转向 LAB2;若位2为1,其它位为0,则转向 LAB3;若位 0至位2都是1,则顺序执行。假定所有的转移都是短程转移。

第五章　汇编语言程序设计

第一节　汇编语言的基本概念

任何计算机都必须在程序控制之下进行有效的工作。为了沟通使用者和计算机之间的信息交换，产生了各种各样的程序设计语言。各种语言都有自己的特点、优势及运行环境，有自己的应用领域和针对性。从使用者的角度看，计算机程序设计语言一般可分为机器语言、汇编语言和高级语言三种不同层次的语言。

所谓汇编语言是一种采用助记符表示的程序设计语言。即用助记符来表示指令的操作码和操作数，用标号或符号代表地址、常量或变量。助记符一般都是英语词的缩写，因而方便人们书写，阅读和检查。一般情况下，一个助记符表示一条机器指令，所以汇编语言也是面向机器的语言。实际上，由汇编语言编写的汇编语言源程序就是机器语言程序的符号表示，汇编语言源程序与其经过汇编所产生的目标代码程序之间有明显的一一对应关系。

用汇编语言编写程序能够直接利用硬件系统的特性(如寄存器、标志、中断系统等)直接对位、字节或字寄存器或存储单元、I/O 端口进行处理，同时也能直接使用 CPU 指令系统和指令系统提供的各种寻址方式，编制出高质量的程序，这样的程序不但占用内存空间少，而且执行速度快。当然，由于源程序和所要解决的问题的数学模型之间的关系不够直观，使得汇编语言程序设计需要较多的软件开发时间，也增加了程序设计过程中出错的可能性。

用汇编语言编写的源程序在交付计算机执行前，也需要翻译成目标程序，机器方能执行。这个翻译过程称为汇编，完成汇编任务的程序称为汇编程序，见图 5-1。

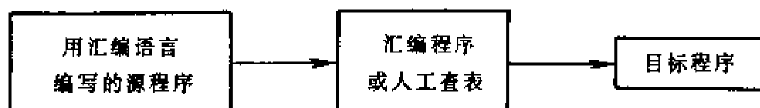


图 5-1　汇编程序的功能示意图

汇编程序是最早也是最成熟的一种系统软件。它除了能够将汇编语言源程序翻译成机器语言程序这一主要功能外，还能够根据用户的要求自动分配存储区域(包括程序区，数据区，暂存区等)；自动地把各种进制数转换成二进制数，把字符转换成 ASCII 码，计算表达式的值等，自动对源程序进行检查，给出错误信息(如非法格式，未定义的助记符，标号，漏掉操作数等)等。具有这些功能的汇编程序又称为基本汇编(或小汇编 ASM)。

在基本汇编的基础上，进一步允许在源程序中把一个指令序列定义为一条宏指令的汇编程序，就叫做宏汇编(MASM)。它包含全部 ASM 功能，还增加了宏指令、结构、记录等高级汇编语言功能。

第二节　汇编语言源程序的格式

在第四章所举出的若干程序段并不是规范的汇编语言程序，下面举出一个比较简单，而比较规范的汇编语言源程序。

例 5.1　要求将两个五字节 16 进制数相加，可以编写出以下汇编语言源程序。

```
DATA    SEGMENT                                 ;定义数据段
        DATA1   DB 0F8H,60H,0ACH,74H,3BH        ;被加数
        DATA2   DB 0C1H,36H,9EH,0D5H,20H        ;加法
DATA    ENDS                                    ;数据段结束
CODE    SEGMMENT                                ;定义代码段
                ASSVME CS:CODE,DS:DATA
        START:  MOV AX,DATA
                MOV   DS,AX                     ;初始化 DS
                MOV   CX,5                      ;循环次数 CX
                MOV   SI,0                      ;置 SI 初值为0
                CLC                             ;清 CF 标志
        LOOPER: MOV   AL,DATA2[SI]              ;取一个字节加数
                ADC   DATA1[SI],AL              ;与被加数相加
                INC   SI                        ;SI 加1
                DEC   CX                        ;CX 减1
                JNZ   LOOPER                    ;若不等于0,转 LOOPER
                HLT                             ;停止
CODE    ENDS                                    ;代码段结束
        END   STAET                             ;源程序结束
```

一、分段结构

由上面的例子可以看出，汇编语言源程序的结构是分段结构形式。一个汇编语言源程序由若干逻辑段(SEGMENT)组成，每个逻辑段以 SEGMENT 语句开始，以 ENDS 语句结束。整个源程序以 END 语句结束。

每个逻辑段内有若干行语句(STATEMENT)，可以说一个汇编源程序是由一行一行的语句组成的。下面我们来讨论汇编语言语句的类型和组成。

二、汇编语言语句的类型和格式

(一)语句的类型

汇编语言源程序中的语句可以分为三种类型：指令语句，伪指令语句和宏指令语句。

1. 指令语句：是能产生目标代码，CPU 可以执行的能完成特定功能的语句，它主要由 CPU 指令组成。

2. 伪指令语句：是一种不产生目标代码的语句，它仅仅在汇编过程中告诉汇编程序应如

何汇编。例如，告诉汇编程序已写出的汇编语言源程序有几个段，段的名字是什么；定义变量，定义过程，给变量分配存储单元，给数字或表达式命名等。所以伪指令语句是为汇编程序在汇编时用的。

3. 宏指令语句：它是一个指令序列。汇编时凡有宏指令语句的地方都将用相应的指令序列的目标代码插入。

(二)语句的格式

指令语句与伪指令语句的格式是类似的，下面主要介绍这两种语句的格式，宏指令语句的格式稍后再作介绍。

一般情况下，汇编语言的语句可以由1～4部分构成：

[名字] 助记符 [操作数] [;注释]

其中带方括号的部分表示任选项，即可以有，也可以没有。如：

```
LOOPER: MOV   AL,DATA[SI]                    ;取一个字节数
DATA1   DB    OF8H,6OH,OACH,74H,3BH          ;定义数组
```

第一条语句是指令语句，其中的"MOV"是 CPU 指令的助记符；第二条语句是伪指令语句，其中的"DB"是伪指令定义符。下面对汇编语言中的各个组成部分进行讨论。

1. 名字

汇编语言语句的第一个组成部分是名字。在指令语句中，这个名字可以是一个标号。指令语句中的标号实质上是指令的符号地址。并非每条指令语句必须有标号，但如果一条指令前面有一个标号，则程序中其它地方就可以引用这个标号。在例5.1中，START、LOOPER 就是标号。标号后面通常有一个冒号。

标号有三种属性：段、偏移量和类型。

标号的段属性是定义标号的程序段的段基值，当程序中引用一个标号时，该标号的段基值应在 CS 寄存器中。

标号的偏移量属性表示标号所在段的起始地址到定义该标号的地址之间的字节数。偏移量是一个16位无符号数。

标号的类型属性有两种：NEAR 和 FAR。前一种标号可以在段内被引用，地址指针为2个字节；后一种标号可以在其它段被引用，地址指针为4个字节。如果定义一个标号时后跟冒号，则汇编程序确认其类型为 NEAR。

伪指令语句中的名字可以是变量名、段名、过程名。与指令语句中的标号不同，这些伪指令语句中的名字并不总是任选的，有些伪指令规定前面必须有名字，有些则不允许有名字，也有一些伪指令的名字是任选的。即不同的伪指令对于是否有名字有不同的规定。伪指令语句的名字后面通常不跟冒号，这是它和标号的一个明显区别。

很多情况下伪指令语句中的名字是变量名，变量名代表存储器中一个数据区的名字，例如例5.1中的 DATA1、DATA2就是变量名。

变量也有三种属性：段、偏移量和类型。

变量的段属性是变量所代表的数据区所在段的段基值。由于数据区一般在存储器的数据段中，因此变量的段基值常常放在 DS 和 ES 寄存器中。

变量的偏移量属性是该变量所在段的起始地址与变量的地址之间的字节数。

变量的类型属性有 BYTE(字节)、WORD(字)、DWORD (双字)、QWORD (四字)、

TBYTE(十字)等,表示数据区中存取操作对象的大小。

2. 助记符

汇编语言语句中的第二个组成部分是助记符。

在指令语句中的第二部分是8086/8088指令系统中指令的助记符,例如:MOV ADC 等等。助记符共计约有90多种。

伪指令语句中的第二部分是伪指令的定义符,例如:DB、SEGMENT、ENDS 、END 、ASSUME 等都是伪指令定义符。它们在程序中的作用是定义变量的类型,定义段以及命令汇编程序结束汇编等,这些操作都是由汇编程序完成的。关于伪指令的作用和使用方法,将在本章第三节进行讨论。

3. 操作数

汇编语言语句中的第三组成部分是操作数。在指令语句中是指令的操作数。可能有单操作数或双操作数,也可能无操作数,而伪指令可能有更多个操作数,当操作数不止一个是,相互之间应该用逗号隔开。

可以作为操作数的有:常量、寄存器、标号、变量和表达式等。

(1)常量

常量就是指令中出现的那些固定值,可以分为数值常量和字符串常量两类。例如,立即数寻址时所用的立即数,直接寻址时所用的地址,ASCII 字符串都是常量,常量除了自身的值以外,没有其它的属性。在源程序中,数值常量可用二进制数、八进制数、十进制数、十六进制数等几种不同表示形式。汇编语言用不同的后缀加以区别。

还应指出,汇编语言中的数值常量的第一位必须是数字,否则汇编时将被看成是标识符。如常数 B7H,在语言中应写成0B7H,FFH 应写成0FFH。

字符串常量是由单引号''括起来的一串字符。例如'ABCDEFG','179'。单引号内的字符在汇编时都以 ASCII 的代码形式存放在存储单元中。如上述两字符串其 ASCII 代码分别为41H,42H,43H,44H,…,47H,31H,37H,39H。字符串最长允许有255个字符。

(2)寄存器

8086/8088CPU 的寄存器可以作为指令的操作数,8086/8088CPU 寄存器有:

8 位寄存器:AH,AL,BH,BL,CH,CL,DH,DL;

16位寄存器:AX,BX,CX,DX,SI,DI,BP,SP,DS,ES,SS,CS。

(3)标号

由于标号代表一条指令的符号地址,因此可以作为转移(无条件转移或条件转移)、过程调用以及循环控制指令的操作数。

(4)变量

因为变量是存储器中某个数据区的名字,因此在指令中可以作为存储器操作数。

(5) 表达式

汇编语言语句中的表达式,按其性质可分为两种:数值表达式和地址表达式。数值表达式产生一个数值结果,只有大小,没有属性。地址表达式的结果不是一个单纯的数值,而是一个表示存储器地址的变量或标号,它有三种属性:段、偏移量和类型。

表达式中常用的运算符有以下几种:

1) 算术运算符

常用的算术运算符有:+(加),-(减),*(乘),/(除)和 MOD(模除,即两个整数相除后

取余数)等。

以上算术运算符可用于数值表达式,运算结果是一个数值。在地址表达式中通常只使用其中的+和-(加和减)两种运算符。

2) 逻辑运算符

逻辑运算符有:AND(逻辑"与"),OR(逻辑"或")、XOR(逻辑"异或")和 NOT(逻辑"非")。

逻辑运算符只用于数值表达式中对数值进行按位逻辑运算,并得到一个数值结果。

3) 关系运算符

关系运算符有:EQ(等于),NE(不等),LT(小于),GT(大于),LE(小于或等于),GE(大于或等于)等。

参与关系运算的必须是两个数值或同一段中的两个存储单元地址,但运算结果只可能是两个特定的数值之一:当关系不成立(假)时,结果为0(全0);当关系成立(真)时,结果为0FFFFH(全1)。例如:

```
MOV   AX,4 EQ 3         ;关系不成立,故(AX)←0
MOV   AX,4 NE 3         ;关系成立,故(AX)←0FFFFH
```

4) 分析运算符

分析运算符可以把存储器操作数分解为它的组成部分,如它的段值,段内偏移量和类型,或取得它所定义的存储空间的大小。分析运算符有 SEG、OFFSET、TYPE、SIZE 和 LENGTH 等。

① SEG 运算符

利用运算符 SEG 可以得到一个标号或变量的段基值。例如下面两条指令将变量 ARRAY 的段基值送 DS 寄存器。

```
MOV   AX,SEG ARRAY
MOV   DS,AX
```

② OFFSET 运算符

利用运算符 OFFSET 可以得到一个标号或变量的偏移地址,例如:

```
MOV   DI,OFFSET DATA1
```

③ TYPE 运算符

运算符 TYPE 的运算结果是一个数值,这个数值与存储器操作数类型属性的对应关系见表5-1。

表5-1 TYPE 返回值与类型的关系

TYPE 返回值	存储器操作数的类型
1	BYTE
2	WORD
4	DWORD
-1	NEAR
-2	FAR

下面是使用 TYPE 运算符的语句例子:

```
VAR     DW      ?                   ;变量VAR的类型为字
ARRAY   DD      10 DUP(?)           ;变量ARRAY的类型为双字
STR     DB      'THIS IS TEST'      ;变量STR的类型为字节
        ⋮
        MOV  AX,TYPE VAR            ;(AX)←2
        MOV  BX,TYPE ARRAY          ;(BX)←4
        MOV  CX,TYPE STR            ;(CX)←1
        ⋮
```

程序中的DW、DD、DB等为伪指令定义符。

④ LENGTH运算符

如果一个变量已用重复操作符DUP说明其变量的个数，则利用LENGTH运算符可得到这个变量的个数。如果未用DUP说明，则得到的结果总是1。

例如上面的例子中已经用"10 DUP(?)"说明变量ARRAY的单元个数，则LENGTH ARRAY的结果为10。

⑤ SIZE运算符

如果一个变量是已用重复操作符DUP说明，则利用SIZE运算符可得到分配给该变量的字节总数。如果未用DUP说明，则得到的结果是TYPE运算的结果。

例如上面例子中变量ARRAY的个数为10，类型为DWORD(双字)，因此，SIZE ARRAY的结果为10×4=40。由此可知，SIZE的运算结果等于LENGTH的运算结果乘以TYPE的运算结果。

SIZE ARRAY=(LENGTH ARRAY)×(TYPE ARRAY)

5) 合成运算符

合成运算符可以用来建立或临时改变变量或标号的类型或存储器操作数的存储单元类型。合成运算符有PTR、THIS、SHORT等。

① PTR运算符

运算符PTR可以指定或修改存储器操作数的类型，例如：

```
INC BYTE PTR[BX][DI]
```

指令中利用PTR运算符明确规定了存储器操作数的类型是BYTE(字节)，因此，本指令将一个8位存储器的内容加1。

利用PTR运算符可以建立一个新的存储器操作数，这个操作数与原来的同名操作数具有相同的段和偏移量，但可以有不同的类型。不过这个新类型只在当前语句中有效。例如：

```
STUFF   DD ?                          ;定义STUFF为双字类型变量
        MOV  BX,WORD PTR STUFF;从STUFF中取一个字到BX
        ⋮
```

② THIS运算符

运算符THIS也可指定存储器操作数的类型。使用THIS运算符可以使标号或变量具有灵活性。例如要求对同一个数据区，既可以字节为单位，又可以字为单位进行存取，则可用以下语句：

```
AREAW   EQU THIS WORD
AREAB   DB 100 DUP(?)
```

上面 AREAW 和 AREAB 实际上代表同一个数据区，其中共有100个字节，但 AREAW 的类型为 WORD，而 AREAB 的类型为 BYTE。

③ SHORT 运算符

运算符 SHORT 指定一个标号的类型为 SHORT(短标号)，即标号到引用该标号之间的距离在－128～＋127个字节的范围内。短标号可以用于转移指令中。使用短标号的指令比使用缺省的近程标号的指令少一个字节。

6）其它运算符

① 段超越运算符“:”

运算符“:”(冒号)跟在段寄存器名(DS,ES,SS 和 CS)之后，表示段超越，用以给一个存储器操作数指定一个段属性，而不管其原来隐含的段是什么。例如：

```
MOV   AX,ES:[SI]
```

② 字节分离运算符 LOW 和 HIGH

运算符 LOW 和 HIGH 分别得到一个数值或地址表达式的低位和高位字节。例如：

```
STUFF       EQU   0ABCDH
            MOV   AH,HIGH STUFF          ;(AH)←0ABH
            MOV   AL,LOW STUFF           ;(AL)←0CDH
```

以上介绍了表达式中使用的各种运算符，如果一个表达式同时具有多个运算符，则按以下规则运算：

① 优先级高的先运算，优先级低的后运算；

② 优先级相同时按表达式中从左到右的顺序运算；

③ 括号可以提高运算的优先级，括号内的运算总是在相邻的运算之前进行。

各种运算符的优先级顺序见表5-2。表中同一行的运算符具有相等的优先级。

表5-2　运算符的优先级

优先级	运　算　符
(高)	
1	LENGTH,SIZE,WIDTH,MASK,(),[],<>
2	·(结构变量名后面的运算符)
3	:(段超越运算符)
4	PTR,OFFSET,SEG,TYPE,THIS
5	HIGH,LOW
6	+,-(一元运算符)
7	*,/,MOD,SHL,SHR
8	+,-(二元运算符)
9	EQ,NE,LT,LE,GT,GE
10	NOT

续表

优先级	运 算 符
11	AND
12	OR,XOR
13	SHORT
(低)	

4. 注释

汇编语言语句的最后一个组成部分是注释。对于一个汇编语言语句来说,注释部分并不是必要的,但是加上适当的注释以后,可以增加源程序的可读性。一个较长的实用程序,如果从头到尾没有任何注释,可能很难读懂。因此,最好在重要的程序段前面以及关键处加上简明扼要的注释。

注释前面要求加上分号";"。如果注释的内容较多,超过一行,则换行以后前面还要加上分号。注释也可以从一行的最前面开始。

汇编程序对于注释不予理会,即注释对汇编后产生的目标程序没有任何影响。

第三节 伪指令语句

伪指令语句中的伪指令,无论表示形式或其在语句中所处的位置,都与 CPU 指令相似。但二者之间有着重要的区别。首先,CPU 指令是给 CPU 的命令,在运行时由 CPU 执行,每条指令对应 CPU 的一种特定的操作。例如传送、加法等;而伪指令是给汇编程序的命令,在汇编过程中由汇编程序进行处理。例如定义数据、分配存储区、定义段以及定义过程等。其次,汇编以后,每条 CPU 指令产生一一对应的目标代码;而伪指令则不产生与之相应的目标代码。

宏汇编程序 MASM 提供了约几十种伪指令,其中有一些伪指令小汇编 ASM 不能支持,例如宏处理伪指令等。根据伪指令的功能,大致可以分为以下几类:

1. 数据定义伪指令
2. 符号定义伪指令
3. 段定义伪指令
4. 过程定义伪指令
5. 宏处理伪指令
6. 模块定义与连接伪指令
7. 处理器方式伪指令
8. 条件伪指令
9. 列表伪指令
10. 其他伪指令

在本教材中,不可能对所有的伪操作逐一进行的说明,本节主要介绍一些常用的基本伪指令。

一、数据定义伪指令

数据定义伪指令的用途是定义一个变量的类型,给存储器赋初值,或者仅仅给变量分配存储单元,而不赋于特定的值。常用的数据定义伪指令有:DB、DW、DD 等。

数据定义伪指令的一般格式为：

[变量名] 伪指令 操作数[,操作数…]

方括号中的变量名为任选项。变量名后面不跟冒号。伪操作后面的操作数可以不止一个。如有多个操作数，互相之间应该用逗号分开。

(一)DB (Define byte)

定义变量的类型为BYTE，给变量分配字节或字节串。DB伪操作后面的操作数每个占有1个字节。

(二)DW (Define word)

定义变量的类型为WORD。DW伪操作后面的操作数每个占有1个字，即2个字节。在内存中存放时，低位字节在前，高位字节在后。

(三)DD (Define double word)

定义变量的类型为DWORD。DD后面的操作数每个占有2个字，即4个字节。在内存中存放时，低位字在前，高位字在后。

数据定义伪操作后面的操作数可以是常数、表达式或字符串，但每项操作数的值不能超过由伪操作所定义的数据类型限定的范围。例如，DB伪指令定义数据的类型为字节，则其范围为无符号数：0～255；带符号数：－128～＋127，等等。字符串必须放在单引号中。另外，超过两个字符的字符串只能用DB伪指令定义。例如：

```
DATA    DB 100,0FFH                      ;存入64H,FFH
EXPR    DB 2*3+7                         ;存入0DH
STR     DB 'WELCOME!'                    ;存入8个字符
AB      DB 'AB'                          ;存入41H,42H
BA      DW 'AB'                          ;存入42H,41H
ABDD    DD 'AB'                          ;存入42H,41H,00H,00H
OFFAB   DW AB                            ;存入变量AB的偏移地址
ADRS    DW TABLE,TABLE+5,TABLE+10        ;存入3个偏移地址
TOTAL   DD TABLE                         ;先存TABLE的偏移地址，
                                          再存段地址
```

以上第一和第二语句中，分别将常数和表达式的值赋于一个变量。第三句的操作数是包含8个字符的字符串(只有DB伪指令才能用)。在第四、五、六句，注意伪指令DB、DW和DD的区别，虽然操作数均为“AB”两个字符，但存入变量的内容各不相同。第七句的操作数是变量AB，而不是字符串，此句将AB的16位偏移地址存入变量OFFAB。第八句存入三个等距的偏移地址，共占6个字节。第九句中的DD伪操作将TABLE的偏移地址和段地址顺序存入变量TOTAL，共占2个字。

除了常数、表达式和字符串外，问号“?”也可以作为数据定义伪指令的操作数，此时仅给变量保留相应的存储单元，而不赋于变量某个确定的初值。

当同样的操作数重复多次时，可用重复操作符“DUP”表示，其形式为：

n DUP(初值 [,初值…])

圆括号中为重复的内容，n为重复次数。如果用“n DUP(?)”作为数据定义伪指令的唯一

操作数，则汇编程序产生一个相应的数据区，但不赋任何初值。重复操作符"DUP"可以嵌套。下面是用问号或"DUP"表示操作数的几个例子。

```
FILLER   DB ?
SUM      DW ?
         DB ?,?,?
BUFFER   DB 10 DUP(?)
ZERO     DW 30 DUP(0)
MASK     DB 5 DUP('OK!')
ARRAY    DB 100 DUP(3 DUP(8),6)
```

其中第一、第二句分别给字节变量FILLER和字变量SUM分配存储单元，但不赋于特定的值。第三句给一个没有名称的字节变量分配3个单元。第四句给变量BUFFER分配10个字节的存储空间。第五句给变量ZERO分配一个数据区，共30个字(即60个字节)，每个字的内容均为零。第六句定义一个数据区，其中有5个重复的字符串"OK!"。共占15个字节。最后一句将变量ARRAY定义为一个数据区，其中包含重复100次的内容：8,8,8,6，共占400个字节。

下面列出几个错误的数据定义伪指令语句。

```
ERR1: DW 99         ;变量名后有冒号
ERR2  DB 25 * 60    ;DB的操作数超过255
ERR3  DD 'ABCD'     ;DD的操作数是超过2个字符的字符串
```

二、符号定义伪指令

符号定义伪指令的用途是给一个符号重新命名，或定义新的类型属性等。上述符号包括汇编语言的变量名、标号名、过程名、寄存器名以及指令助记符等。

常用的符号定义伪指令有：EQU、=(等号)和LABLE。

(一)EQU

格式：

名字 EQU 表达式

EQU伪指令将表达式的值赋予一个名字。以后可用这个名字来代替上述表达式。格式中的表达式可以是一个常数，符号，数值表达式或地址表达式等。例如：

```
CR   EQU 0DH                 ;常数
LF   EQU 0AH
A    EQU ASCII-TABLE         ;变量
STR  EQU 64 * 1024           ;数值表达式
ADR  EQU ES:[BP+DI+5]        ;地址表达式
CBD  EQU AAM                 ;指令助记符
```

利用EQU伪指令，可以用一个名字代表一个数值，或用一个较简短的名字来代替一个较长的名字。

如果源程序中需要多次引用某一表达式，则可以利用EQU伪操作给其赋一个名字，以代替程序中的表达式，从而使程序更加简洁，便于阅读。将来如果改变表达式的值，也只需修改一

处，而不必修改多处，使程序易于维护。需要注意一个问题，EQU 伪指令不允许对同一符号重复定义。

（二）＝（等号）

格式：

名字＝表达式

“＝”（等号）伪指令的功能与 EQU 伪指令基本相同，主要区别在于它可以对同一个名字重复定义。例如

```
COUNT=10
      MOV   CX,COUNT              ;(CX)←10
        ⋮
COUNT=COUNT-1
      MOV   BX,COUNT              ;(BX)←9
        ⋮
```

（三）LABLE

LABLE 伪指令的用途是定义标号或变量的类型。格式：

名字 LABLE 类型

变量的类型可以是 BYTE、WORD、DWORD。标号的类型可以是 NEAR 或 FAR。

利用 LABEL 伪指令可以使同一个数据区兼有 BYTE 和 WORD 两种属性，这样，在以后的程序中可根据不同的需要分别以字节为单位，或以字为单位存取其中的数据，例如：

```
AREAW   LABEL WORD                 ;变量 AREAW 类型为 WORD
AREAB   DB 100 DUP(?)              ;变量 AREAB 类型为 BYTE
          ⋮
        MOV   AREAW,AX             ;AX 送第1,第2字节中
          ⋮
        MOV   AREAB[49],AL         ;AL 送第50字节中
```

LABLE 伪指令也可以将一个属性已经定义为 NEAR，或者后面跟有冒号（隐含属性为 NEAR）的标号再定义为 FAR。例如：

```
AGAINF    LABEL    FAR    ;定义标号 AGAINF 的属性为 FAR
AGAIN:    PUSH     AX     ;标号 AHAIN 的属性为 NEAR
```

上面的过程既可以用标号 AGAIN 在本段内被调用，也可以利用标号 AGAINF 被其他段调用。

三、段定义伪指令

段定义伪指令的用途是在汇编语言源程序中定义逻辑段，常用的段定义伪指令有 ASSUME 和 SEGMENT、ENDS 等 。

（二）SEGMENT/ENDS

格式：

段名 SEGMENT [定位类型] [组合类型] ['类别']

⋮

段名 ENDS

SEGMENT 伪指令用于定义一个逻辑段，给逻辑段赋予一个段名，并以后面的任选项(定位类型、组合类型、'类别')规定该逻辑段的其他特性。SEGMENT 伪指令位于一个逻辑段的开始，而 ENDS 伪指令则表示一个逻辑段的结束。这两个伪操作总是成对出现，二者前面的段名必须一致。两个语句之间的部分即是该逻辑段的内容。例如，对于代码段，其中主要有 CPU 指令及其他伪指令。对于数据段和附加段，主要有定义数据区的伪指令，等等。一个源程序中不同逻辑段的段名可以各不相同，但也允许相同。

SEGMENT 伪指令后面还有三个任选项，在上面的格式中，它们都放在方括号内，表示可选择。如果有，三者的顺序必须符合格式中的规定。这些任选项是给汇编程序和连接程序(LINK)的命令。

SEGNENT 伪指令后面的任选项告诉汇编程序和连接程序，如何确定段的边界，以及如何组合几个不同的段等等。下面分别讨论。

1. 定位类型(ALIGN)

定位类型任选项告诉汇编程序如何确定逻辑段的边界在存储器中的位置。定位类型共有以下四种：

(1)BYTE

表示逻辑段从字节的边界开始，即可以从任何地址开始。此时本段的起始地址紧接在前一个段的后面。

(2)WORD

表示逻辑段从字的边界开始。2个字节为1个字，此时本段的的起始地址必须是偶数。

(3)PARA

表示逻辑段从一个节(PARAGRAPH)的边界开始。通常16个字节称为一个节，故本段的开始地址(十六进制)应为××××0H。如果省略定位类型任选项，则默认其为 PARA。

(4)PAGE

表示逻辑段从页边界开始。通常256个字节称为一页，故本段的起始地址(十六进制)应为×××00H。

例5.2　SEGMENT 伪操作的定位类型应用举例。

```
STACK    SEGMENT STACK             ;STACK 段，定位类型无
           DB 100 DUP(?)           ;长度为100字节
STACK    ENDS                      ;STACK 段结束
DATA1    SEGMINT BYTE              ;DATA1段，定位类型 BYTE
STRING     DB 'This is an example!' ;长度为19字节
DATA1    ENDS                      ;DTAT1段结束
DATA2    SEGMINT WORD              ;DATA2段，定位类型 WORD
BUFFER   DW 40 DUP(0)              ;长度为40个字，即80字节
DATA2    ENDS                      ;DATA2段结束
CODE1    SEGMINT PAGE              ;CODE1段，定位类型 PAGE
```

```
        ⋮
CODE1   ENDS                    ;CODE1段结束
CODE2   SEGMENT                 ;CODE2段,定位类型无
            ⋮
 START:     MOV  AX,STACK
            MOV  SS,AX
            ⋮
CODE2   ENDS                    ;CODE2段结束
        END  START              ;源程序结束
```

本例的源程序中共有五个逻辑段,它们的段名和定位类型分别为:

STACK 段	PARA
DATA1段	BYTE
DATA2段	WORD
CODE1段	PAGE
CODE2段	PARA

已经知道其中STACK段的长度为100字节(64H),DATA1段的长度为19字节(13H),DATA2段的长度为40个字,即80字节(50H)。假设CODE1段占用13字节(0DH),CODE2段占用52字节(34H)。

如果将以上进行汇编和连接,然后再来观察各逻辑段的目标代码或数据装入存储器的情况。由表5-3可清楚地看出,当SEGMENT伪指令的定位类型不同时,对段起始边界规定也不相同。

表5-3　例5-2各逻辑段的起始地址和结束地址

段 名	定位类型	字节数	起始地址	结束地址
STACK	PARA	100(64H)	00000H	00063H
DATA1	BYTE	19(13H)	00064H	00076H
DATA2	WORD	80(50H)	00078H	000C7H
CODE1	PAGE	13(ODH)	00100H	0010CH
CODE2	PARA	52(34H)	00110H	00143H

2. 组合类型(Combine)

SEGMENT伪指令的第二个任选项是组合类型,它告诉汇编程序,当装入存储器时各个逻辑段如何进行组合。组合类型共有六种:

(1)不组合

如果SEGMENT伪指令的组合类型任选项缺省,则汇编程序认为这个逻辑段是不组合的。也就是说,不同程序中的逻辑段,即使具有相同的段名,也分别作为不同的逻辑段装入内存,不进行组合。

但是,对于组合类型任选项缺省的同名逻辑段,如果属于同一个程序模块,则被集中成为一个逻辑段。

(2)PUBLIC

连接时,对于不同程序模块中的逻辑段,只要具有相同的段名,就把这些段集中成为一个

逻辑段装入内存。

(3)STACK

组合类型为 STACK 时,其含意与 PUBLIC 基本一样。即不同程序中的逻辑段，如果段名相同,则集中成为一个逻辑段。不过组合类型 STACK 仅限于作为堆栈区域的逻辑段使用。顺便提一下,在执行程序(.EXE)中,堆栈指针 SP 设置在这个集中以后的堆栈段(最终地址+1)处。

(4)COMMON

连接时,对于不同程序中的逻辑段,如果具有相同的段名,则都从同一个地址开始装入,因而各个逻辑段将发生重叠。最后,连接以后的段的长度等于原来最长的逻辑段的长度,重叠部分的内容是最后一个逻辑段的内容。

(5)MEMORY

表示当几个逻辑段连接时,本逻辑段定位在地址最高的地方。如果被连接的逻辑段中有多个段的组合类型都是 MEMORY,则汇编程序只将首先遇到的段作为 MEMORY 段,而其余的段均当作 COMMON 段处理。

(6)AT 表达式

这种组合类型表示本逻辑段根据表达式求值的结果定位段地址。例如 AT 8A00H,表示本段的段地址为8A00H,则本段从存储器的物理地址8A000H 开始装入。

3.'类型'('Class')

SEGMENT 伪指令的第三个任选项是类别,类别必须放在单引号内。类别的作用是在连接时决定各逻辑段的装入顺序。当几个程序模块进行连接时,其中具有相同类别名的逻辑段被装入连续的内存区,类别名相同的逻辑段,按出现的先后顺序排列。没有类别名的逻辑段,与其他无类别名的逻辑段一起连续装入内存。

例如,假设一个主程序中有五个逻辑段,段名和类别名分别为:

STK1段—'STACK'

CODE1段—无

DATA1段—'BUFFER'

DATA2段—'TABLE'

DATA3段—'BUFFER'

还有一个子程序,包括四个逻辑段,段名和类型名分别为:

DATA4段—'TABLE'

DATA5段—'BUFFER'

STK2段—'STACK'

CODE2段—无

当将上述主程序和子程序进行连接时,两个程序模块中各逻辑段装入内存的顺序见图5-2。

(二) ASSUME

格式:

ASSUME 段寄存器名:段名[,段寄存器名:段名[,…]]

对于8086/8088 CPU 而言,以上格式中的段寄存器名可以是 CS、DS、ES 或 SS。段名可以

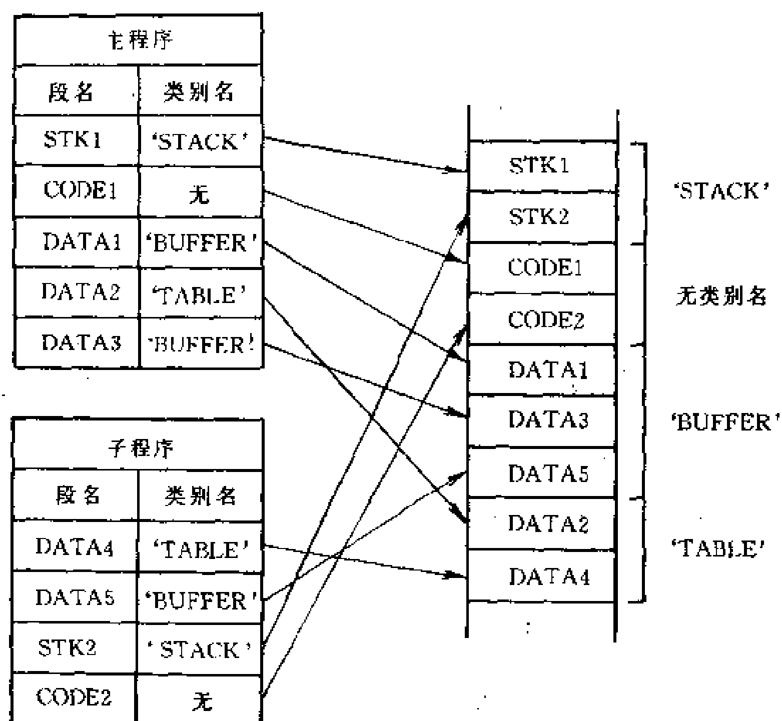


图5-2　逻辑段按类别装入内存的示意图

是曾用 SEGMENT 伪操作定义过的某一个段名或者组名，以及在一个标号或变量前面加上分析运算符 SEG 所构成的表达式，还可以是关键字 NOTHING。

ASSUME 伪指令告诉汇编程序，将某一个段寄存器设置为某一个逻辑段的段址，即明确指出源程序中的逻辑段与物理段之间的关系。当汇编程序汇编一个逻辑段时，即可利用相应的段寄存器寻址该逻辑段中的指令或数据。关键字 NOTHING 表示取消前面用 ASSUME 伪指令对这个段寄存器的设置。在一个源程序中，ASSUME 伪操作应该放在可执行程序开始位置的前面。还需指出一点，ASSUME 伪指令只是通知汇编程序有关段寄存器与逻辑段的关系，并没有给段寄存器赋于实际的初值。例如：

```
CODE SEGMENT
        ASSUME  CS:CODE,DS:DATA1,SS:STACK
        MOV   AX,DATA1
        MOV   DS,AX
        MOV   AX,STACK
        MOV   SS,AX
          ⋮
CODE   ENDS
```

四、过程定义伪指令

(一) PROC/ENDP

格式：

过程名 PROC [NEAR]/FAR

```
        ⋮
        RET
        ⋮
过程名 ENDP
```

其中 PROC 伪指令定义一个过程，赋于过程一个名字，并指出该过程的类型属性为 NEAR 或 FAR。如果没有特别指明类型，则认为过程的类型是 NEAR。伪指令 ENDP 标志过程的结束。上述两个伪指令前面的过程名必须一致。

当一个程序段被定义为过程后，程序中其他地方就可以用 CALL 指令调用这个过程。调用一个过程的格式为：

```
CALL 过程名
```

过程名实质上是过程入口的符号地址，它和标号一样，也有三种属性：段、偏移量和类型。过程的类型属性可以是 NEAR 或 FAR。

一般来说，被定义为过程的程序段中应该有返回指令 RET，但不一定是最后一条指令，也可以有不止一条 RET 指令。执行 RET 指令后，控制返回到原来调用指令的下一条指令。

过程的定义和调用均可嵌套。例如：

```
NAME1   PROC   FAR
          ⋮
        CALL   NAME2
          ⋮
        RET
NAME2   PROC   NEAR
          ⋮
        RET
NANE2   ENDP
NAME1   ENDP
```

上面过程 NAME1的定义中包含着另一个过程 NAME2的定义。NAME1 本身是一个可以被调用的过程，而它也可以再调用其他的过程。

五、模块定义与连接伪指令

在编写规模比较大的汇编语言程序时，可以将整个程序划分成为几个独立的源程序(或称模块)，然后将各个模块分别进行汇编，生成各自的目标程序，最后将它们连接成为一个完整的可执行程序。各个模块之间可以相互进行符号访问。也就是说，在一个模块中定义的符号可以被另一个模块引用。通常称这类符号为外部符号，而将那些在一个模块中定义，只在同一模块中引用的符号称为局部符号。

为了进行连接和在这些将要连接在一起的模块之间实现互相的符号访问，以便进行变量传送，常常使用以下几个伪指令：NAME、END、PUBLIC 和 EXTRN。

(一) NAME

NAME 伪指令用于给源程序汇编以后得到的目标程序指定一个模块名，连接时需要使用

这个目标程序的模块名。其格式为

NAME 模块名

NAME 的前面不允许再加上标号，例如下面的表示方式是非法的：

BEGIN：NAME MODNAME

如果程序中没有 NAME 伪指令，则汇编程序将 TITLE 伪指令(TITLE 属于列表伪指令)后面“标题名”中的前六个字符作为模块名。如果源程序中既没有使用 NAME，也没有使用 TITLE 伪指令，则汇编程序将源程序的文件名作为目标程序的模块名。

(二) END

END 伪指令表示源程序到此结束，指示汇编程序停止汇编，对于 END 后面的语句可以不予理会。格式为

END [标号]

END 伪指令后面的标号表示程序执行的启动地址。END 伪指令将标号的段基值和偏移地址分别提供给 CS 和 IP 寄存器。方括号中的标号是任选项。如果有多个模块连接在一起，则只有主模块的 END 语句使用标号。

(三) PUBLIC

PUBLIC 伪指令说明本模块中的某些符号是公共的，即这些符号可以提供给将被连接在一起的其他模块使用。格式为

PUBLIC 符号[,…]

其中的符号可以是本模块中定义的变量、标号或数值的名字，包括用 PROC 伪指令定义的过程名等。PUBLIC 伪指令可以安排在源程序的任何地方。

(四) EXTRN

EXTRN 伪指令说明本模块中所用的某些符号是外部的，即这些符号在将被连接在一起的其他模块中定义(在定义这些符号的模块中还必须用 PUBLIC 伪指令说明)。格式为

EXTRN 名字:类型[,…]

其中的名字必须是其他模块中定义的符号；上述格式中的类型必须与定义这些符号的模块中的类型说明一致。如为变量，类型可以是 BYTE、WORD 或 DWORD 等；如为标号和过程，类型可以是 NEAR 或 FAR；如果是数值，类型可以是 ABS，等等。

第四节　宏指令语句

在汇编语言中，如果在源程序中需要多次使用同一个程序段，可以将这个程序段定义为一个宏指令，然后每次需要时，即可简单地用宏指令名来代替(称为宏调用)，从而避免了重复书写，使源程序更加简洁、易读。

Microsoft 宏汇编程序 MASM 提供了丰富的宏操作伪指令，但是小汇编 ASM 不具备宏处理功能。下面讨论几种常用的宏指令。

一、MACRO/ENDM

格式：

```
宏指令名 MACRO
        ⋮(宏定义体)
        ENDM
```

MACRO 伪指令是宏定义符，它将一个宏指令名定义为宏定义体中包含的程序段。ENDM 表示宏定义结束，前面不要加上宏指令名。进行一次宏定义，以后就可以多次用宏指令名进行宏调用。但是必须先定义，后调用。

当汇编时，MASM 对每个宏指令名，自动用相应宏定义体中的程序段代替，这个过程称为宏扩展。总之，使用宏的过程共有三步：首先进行宏定义；然后可以进行宏调用；最后，汇编时由 MASM 进行宏扩展。

宏定义允许嵌套，即宏定义体中可以包含另一个宏定义，而且宏定义体中也可以有宏调用，但是也必须先定义后调用。

请看几个宏定义的例子。

例5.3　若源程序中多处需要将 BL 和 CL 寄存器中两位压缩的 BCD 数相加，并将和送回 BL 寄存器，则可象下述这样定义宏指令，然后在需要的地方进行调用。

```
DECADD  MACRO
        ADD   BL,CL
        DAA
        ENDM
```

宏定义伪操作允许带参数，此时可定义的宏指令具有较强的通用性。带参数的宏定义格式如下所示：

```
宏指令名 MACRO 参数[,…]
        ⋮(宏定义体)
        ENDM
```

以上宏定义中的参数称为形式参数(dummy parameter)，或称哑元。当形式参数不止一个时，互相之间要用逗号分开。以后宏调用时，应在宏指令名后面写上相应的实际参数(actual parameter)，或称实元。一般情况下，实际参数与形式参数的个数和顺序均为一一对应。但是，汇编程序允许二者的个数不等。当实际参数多于形式参数时，多余的实际参数被忽略；当形式参数多于实际参数时，认为多余的形式参数为空。

例5.4

```
DECADD1  MACRO OPR1,OPR2
         MOV   AL,OPR1
         ADD   AL,OPR2
         DAA
         MOV   OPR1,AL
         ENDM
```

利用本例中的宏定义可对分别存放在任何8位寄存器或存储单元中的两个压缩BCD数进行加法运算。例如有以下宏调用：

```
DECADD1 DL,BUFFER
DECADD1 AREA1,AREA2
```

则汇编时进行宏扩展，得到以下指令：

```
DECADD1 DL,BUFFER
+ MOV   AL,DL
+ ADD   AL,BUFFER
+ DAA
+ MOV   DL,AL
DECADD1 AREA1,AREA2
+ MOV   AL,AREA1
+ ADD   AL,AREA2
+ DAA
+ MOV   AREA1,AL
```

宏扩展后，原来宏定义体中的指令前面加上了符号"+"，以资区别。

宏定义中的形式参数不仅可以是指令的操作数，而且可以是指令的操作码，甚至可以是操作码或操作数的一部分，此时在宏定义体中必须用宏操作符"&"将形式参数和其他字符分隔开。宏扩展时，"&"前后的实际参数和其他字符紧接在一起，而符号"&"不再出现。

二、PURGE

伪指令 PURGE 的用途是取消已有的宏定义。格式：

PURGE 宏指令名[,…]

宏汇编程序 MASM 允许所定义的宏指令名与 CPU 指令的助记符或伪指令的名字相同，此时，MASM 优先考虑宏指令的定义，而与宏指令同名的指令助记符或伪指令原来的含义失效。当用 PURGE 伪指令取消上述宏指令的定义后，即可恢复这些 CPU 指令或伪指令原来的含义。另外，如欲对一个宏指令名重新定义，则必须先用 PURGE 伪指令取消以前的定义，然后再重新定义。

三、宏指令与子程序的区别

宏指令是用一条指令来代替一段程序以简化程序，而子程序(过程)也有类似的功能。那么这两者之间有什么区别?

1. 宏指令由宏汇编程序 MASM 在汇编过程中进行处理，在每个宏调用处，将相应的宏定义体插入；而调用指令 CALL 和返回指令 RET 则是 CPU 指令。执行 CALL 指令时，CPU 使程序的控制转移到子程序的入口地址。

2. 宏指令简化了源程序，但不能简化目标程序，汇编以后，在宏定义处不产生机器代码。但在每个宏调用处，通过宏扩展，重复程序段的机器代码仍然出现多次，因此不节省内存单元。对于子程序来说，在目标程序中定义子程序的地方将产生相应的机器代码，但每次调用时，只

需用 CALL 指令，不再重复出现子程序的机器代码，一般来说可以节省内存单元。

3. 从执行时间来看，子程序调用和返回需要保护断点、恢复断点等等，都将额外占用 CPU 的时间，而宏指令则不需要，因此相对来说，执行速度较快。

此外，宏指令更加接近高级语言，而且传送参数更加方便。

第五节 汇编语言程序的上机过程

在计算机上建立和运行汇编语言时，首先要用编辑程序（如行编辑程序 EDLIN 或全屏幕编辑程序 WPS 等）建立汇编语言源程序（其扩展名必须为. ASM），源程序就是用汇编语言的语句编写的程序。汇编语言源程序是不能直接被计算机运行的，必须经过汇编程序（MASM 或 ASM）加以汇编（翻译），把源程序文件转换成为用机器码（二进制代码）表示的目标程序文件（其扩展名为. OBJ）。若在汇编过程中没有出现语法错误，则汇编结束后，还必须经过连接程序（LINK）把目标程序文件与库文件或其它目标文件连接在一起形成可执行文件（其扩展名为. EXE文件）。这时就可以在 DOS 下直接键入文件名运行此程序。

因此，在计算机上运行汇编语言程序的步骤是：

① 用编辑程序建立 ASM 源程序文件·

② 用汇编程序（MASM 或 ASM）把 ASM 文件汇编成 OBJ 文件；

③ 用连接程序（LINK）把 OBJ 文件转换成 EXE 文件；

④ 在 DOS 命令状态下直接键入文件名就可执行该文件。

上述上机过程可用图5-3表示：

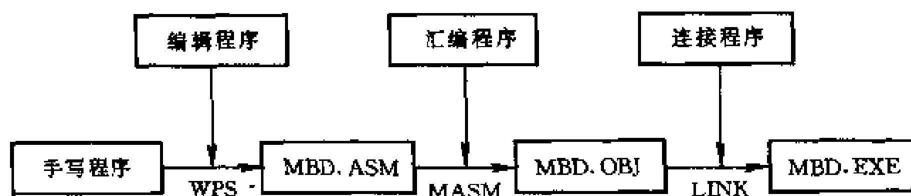


图5-3 汇编语言程序上机过程

一、用编辑程序建立汇编语言源程序文件（ASM 文件）

例如要编写一个多字节相加的汇编语言源程序，我们可以用字处理程序 WPS 在磁盘上建立源程序文件。调用 WPS，然后使用编辑非文书文件方式（即选 N 命令）建立以MBA. ASM 为文件名的源程序文件，输入如下的源程序：

```
DATA   SEGMENT                              ;定义数据段
       ARRY1  DB 1,2,3,5,6,7,9,10,11
       ARRY2  DB 21,17,35,15,50,26,41,42,28
       ARRY3  DB 9 DUP(?)
       N   DW 9
DATA   ENDS                                 ;数据段结束
STACK  SEGMENT PARA STACK 'STACK';定义堆栈段
       DW   100 DUP(?)
```

```
STACK ENDS                                    ;堆栈段结束
CODE   SEGMENT                                ;定义代码段
       ASSUME  CS:CODE,DS:DATA,SS:STACK
MAIN   PROC  FAR                              ;主程序部分
START:MOV   AX,STACK                          ;把堆栈段地址送 AX
       MOV   SS,AX                            ;然后通过 AX 送入 SS
       PUSH  DS
       MOV   AX,0
       PUSH  AX
       MOV   AX,DATA                          ;把数据段地址送 AX
       MOV   DS,AX                            ;然后通过 AX 送入 DS
       MOV   SI,OFFSET ARRY1                  ;把 ARRY1的偏移量地址送入 SI
       MOV   DI,OFFSET ARRY2                  ;把 ARRY2的偏移量地址送入 DI
       MOV   CX,N
       CALL  ADDFA                            ;调用子程序 ADDFA
       RET                                    ;返回 DOS
MAIN   ENDP                                   ;主程序结束
ADDFA PROC  NEAR                              ;定义子程序
       PUSH  AX
       PUSH  CX
       PUSH  SI
       PUSH  DI
       MOV   BX,OFFSET ARRY3
LOOP1: MOV   AL,[SI]
       ADD   AL,[DI]
       MOV [BX],AL
       INC   SI
       INC   DI
       INC   BX
       LOOP  LOOP1
       POP   DI
       POP   SI
       POP   CX
       POP   AX
       RET
ADDFA ENDP                                    ;子程序结束
CODE   ENDS                                   ;代码段结束
       END   MAIN                             ;汇编结束
```

WPS 的使用方法可查阅手册。如果没有 WPS 则可使用 TC 编辑程序或其他编辑程序(如 DOS 的 EDIT)以及其它字处理程序。

二、用汇编程序将ASM文件汇编成目标程序文件(OBJ文件)

在对源程序文件(简称 ASM 文件)汇编时,汇编程序将对 ASM 文件进行二遍扫描。若程序文件中有语法错误,则结束汇编后,将指出源程序中的错误,这时用编辑程序修改源程序中的错误,再经过汇编,直到最后得到无错误的目标程序,即 OBJ 文件。

因此,汇编程序的主要功能可以概括为以下三点:

(1) 检查源程序中的语法错误,并给出错误信息。

(2) 产生目标程序文件(OBJ 文件)。

(3) 展开宏指令。

完成汇编功能的是汇编程序 ASM 或宏汇编程序 MASM。MASM 比 ASM 的功能强,但 MASM 需要占据较大的内存空间,当内存空间较小时(如64K),只能使用 ASM。

汇编过程:

当源程序建立以后,仍以 MBA. ASM 程序为例,我们用汇编程序 MASM 对 MBA. ASM 源程序文件进行汇编,以便产生机器码的目标程序文件 MBA. OBJ,其操作步骤如下:

```
C>MASM MBA
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985,1987,All rights reserved

Object filename [MBA.OBJ]:
Source listing [NUL.LET]:MBA
Cross-reference [NUL.CRF]:MBA

50468 + 303948 Bytes symbol space free

     0 Warning Errors
     0 Severe Errors
```

由此可知,汇编程序调入后,首先显示版本号,然后出现三个提示行。第一个提示行为:

```
Object filename [MBA.OBJ]:
```

这是询问目标程序文件名,方括号内为机器规定的默认的文件名,通常直接打回车,表示采用默认的文件名,如上所示,这是我们汇编的主要目的。第二个提示为:

```
Source listing [NUL.LST]:
```

这是询问是否建立列表文件,若不建立,直接打回车;若要建立,则打入文件名再回车,如上所示,这里是 MBA。列表文件中同时列出源程序和机器语言程序清单,并给出符号表,有利于程序调试。第三个提示行为:

```
Cross-reference [NUL.CRF]:
```

这是询问是否要建立交叉索引文件,若不要建立,则直接回车;若要建立,则应打入文件名,这就建立了 MBA. CRF 文件,为了建立交叉索引文件,必须调用 CREF. EXE 程序,即打入:

```
C>CREF MBA
```

```
Microsoft (R) Cross-Reference Utility Version 5.00
Copyright (C) Microsoft Corp 1981-1985,1987. All rights reserved.
listing [MBA.REF]:

11 Symbols
```

这时首先显示版本号,然后出现一个提示行:

```
Listing [MBA.REF]:
```

这是询问交叉索引文件名,这时可用回车承认方括号内机器默认的文件名,如上所示。这样就建立了 MBA.REF 文件。其内容是用户定义的所有符号(包括变量),并给出每个符号定义所在的行号(附以#)以及引用的行号。

调入汇编程序,当我们回答了上述各提问行的询问之后,汇编程序就对源程序进行汇编。若汇编过程中发现源程序有语法错误,则列出有错误的语句和错误的代码。错误分警告错误(Wraning Errors)和严重错误(Severe Errors)。警告错误是指汇编程序认为的一般性错误;严重错误是指汇编程序认为无法进行正确汇编的错误,并给出错误的个数、错误的性质。这时,就要对错误进行分析,找出问题和原因,然后再调用编辑程序加以修改,修改后重新汇编,直到汇编后无错误为止。

三、用连接程序生成可执行程序文件(EXE 文件)

经汇编后产生的目标程序文件(OBJ 文件)并不是可执行程序文件(EXE 文件),必须经连接以后,才能成为可执行文件。连接程序并不是专为汇编语言程序设计的,如果一个程序是由若干个模块组成的,也可通过连接程序 LINK 把它们连接在一起,这些模块可以是汇编程序产生的目标文件,也可以是高级语言编译程序产生的目标文件。

连接过程如下:

```
C>LINK MBA
Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

Run File [MBA.EXE]:
List File [NUL.MAP]:MBA
Libraries [.LIB]:
```

由此可知,在连接程序调入后,首先显示版本号,然后出现三个提示行:

```
Run File [MBA.EXE]:
```

这是询问要产生的可执行文件的文件名,一般直接回车采用方括号内规定的隐含文件名。

```
List File [NUL.MAP]:
```

这是询问是否要建立连接映象文件。若不要建立,则直接回车;若要建立,则打入文件名再回车。我们这里是要建立该文件,则打入文件名 MBA。

```
Libraries [.LIB]:
```

这是询问是否用到库文件,若无特殊需要,则直接打入回车即可。

上述提示行回答后，连接程序开始连接，若连接过程中有错，则显示错误信息，错误分析清楚后，要重新调入编辑程序进行修改，然后重新汇编，再经过连接，直至无错为止。连接以后，便产生了可执行程序文件(.EXE 文件)。

四、程序的执行

当我们建立了可执行文件 MBA.EXE 后，就可直接在 DOS 下执行该程序。键入：

```
C>MBA
C>
```

因为在程序中没有用到输出指令，故程序执行后在看不到执行的结果。

五、汇编语言和操作系统 PC-DOS 的接口

当我们编写的汇编语言源程序是在 PC-DOS 环境下运行时，必须了解汇编语言是如何同操作系统接口的。

当我们用连接程序对其进行连接和定位时，操作系统为每一个用户程序建立了一程序段前缀区 PSP，其长度为256个字节，主要用于存放所要执行程序的有关信息，同时也提供了程序和操作系统的接口。操作系统在程序段前缀的开始处(偏移地址0000H)安排了一条 INT 20H 软中断指令。INT 20H 中断服务程序由 PC-DOS 提供，执行该服务程序后，控制就转移到 DOS，即返回到 DOS 管理的状态。因此，用户在组织程序时，必须使程序执行完后能去执行存放于 PSP 开始处的 INT 20H 指令，这样便返回到 DOS，否则就无法继续键入命令和程序。

PC-DOS 在建立了程序段前缀区 PSP 之后，就将要执行的程序从磁盘装入内存。在定位程序时，DOS 将代码段置于 PSP 下方，代码段之后是数据段，最后放置堆栈段。内存分配好之后，DOS 就设置段寄存器 DS 和 ES 的值，以使它们指向 PSP 的开始处，即 INT 20H 的存放地址，同时将 CS 设置为 PSP 后面代码段的段基值，IP 设置为指向代码段中第一条要执行的指令位置，把 SS 设置为指向堆栈的段基值，让 SP 指向堆栈段的栈底(取决于堆栈的长度)，然后系统开始执行用户程序。

为了保证用户程序执行完后，能回到 DOS，可使用如下两种方法：

1. 标准方法

首先将用户程序的主程序定义成一个 FAR 过程，其最后一条指令为 RET。然后在代码段的主程序(即 FAR 过程)的开始部分用如下三条指令将 PSP 中 INT 20H 指令的段基值及偏移地址压入堆栈：

```
PUSH  DS            ;保护 PSP 段地址
MOV   AX,0          ;保护偏移地址0
PUSH  AX
```

这样，当程序执行到主程序的最后一条指令 RET 时，由于该过程具有 FAR 属性，故存在堆栈内的两个字就分别弹出到 CS 和 IP，便执行 INT 20H 指令，使控制返回到 DOS 状态。

此外，由于开始执行用户程序时，DS 并不设置在用户的数据段的起始处，ES 也同样不设置在用户的附加段起始处，因而在主程序开始处，继上述三条指令之后，应该重新装填 DS 和 ES 的值。

2. 非标准方法

也可在用户的程序中不定义过程段，只在代码段结束之前(即 CODE ENDS 之前)，增加两条语句：

```
MOV   AH,4CH
INT   21H
```

则程序执行完后也会自动返回 DOS 状态。

第六节 汇编语言程序设计的基本方法

一、汇编语言程序设计的基本过程

汇编语言程序设计的基本过程可分为以下几个步骤：

(一)分析问题，明确要求

分析问题就是深入实际，对所要解决的问题进行全面了解和分析。一个实际问题往往是比较复杂的，在深入分析的基础上，要善于抓住主要矛盾，剔除次要矛盾，抽取问题的本质。

明确要求就是明确用户的要求，依据给出的条件和数据，对需要进行哪些处理，输出什么样的结果，进行可行性分析。

(二)建立数学模型

在分析问题和明确要求的基础上，要建立数学模型，将一个物理过程或工作状态用数学形式表达出来。

(三)确定算法和处理方案

数学模型建立后，必须研究和确定算法：所谓算法是指解决某些问题的计算方法，不同类的问题有不同的计算方法。根据问题的特点，对计算方法进行优化。若没有现成方法可用，必须通过实践摸索，并总结出算法思想和规律性。

(四)画流程图

流程图是程序算法的图形描述，它以图形的方式把解决问题的先后次序和程序的逻辑结构直观地、形象地描述出来，使解题的思路清析，有利于理解、阅读和编制程序，还有利于调试、修改程序和减少错误等。

(五)编制程序

在编制程序时，应当先分配好存储空间和工作单元及 CPU 内部的寄存器，然后根据流程图和确定的算法逐条语句编写程序。

(六)上机调试

程序编好后，必须上机调试，特别是对于复杂的问题，往往要分解成若干个子问题，分别由几个人编写，而形成若干个程序模块，把它们组装在一起，才能形成总体程序。一般来说，总会有这样或那样的问题或错误，这些问题和错误在调试程序时通常都可以发现，然后进行修改，再调试，再修改，直到所有的问题解决为止。

(七)试运行和分析结果

试运行和分析结果是为了检验程序是否达到了设计要求，是否满足用户提出的需求，所确定的设计方案是否可行。若没有达到设计要求，不满足用户的需求，就必须从分析问题开始检查修正原有的设计方案，直到符合设计要求和满足用户需求为止。

（六）整理资料，投入运行

在试运行满足要求之后，应当系统地整理材料，有关资料要及时提交用户，以便正常投入运行。

二、程序结构化的概念

在计算机发展的初期，由于计算机硬件贵、内存容量小和运算速度低，因此，要求程序运行时间尽可能短，占用内存尽可能少。就是说，当时衡量程序质量好坏的主要标准是占用内存的大小和运算时间的长短。为了达到这一目的，人们挖空心思寻找技巧，这种程序人们很难理解和消化，造成人力和时间的严重浪费，而且程序实际没有统一的规范，弊端大。

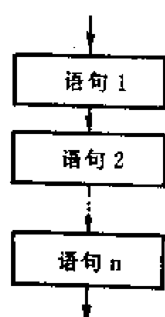


图5-4　顺序结构

随着计算机的迅速发展，特别是大规模和超大规模集成电路技术的兴起，使计算机硬件价格大大下降，内存容量不断扩大，运算速度大幅度提高。因此，运行时间短和节省内存已不是主要矛盾，而应使程序具有良好的结构、清晰的层次、容易阅读和理解、容易修改和查错，这就对以前的传统设计方法提出了挑战，从而产生了结构化程序设计方法。

结构化程序设计是1969年由荷兰学者 E. W. Dijkstra 等人提出的。这种新的程序设计方法包括以下三方面的内容：

（一）程序由一些基本结构组成。这些基本结构包括顺序结构、分支结构和循环结构，由这些基本结构构成一个结构化程序。

1. 顺序结构

顺序结构是按照语句实现的先后次序执行一系列的操作，它没有分支、循环和转移，如图5-4所示。

顺序结构的程序一般是简单程序。

2. 分支结构

分支结构根据不同情况做出判断和选择，以便执行不同的程序段。分支的意思是在两个不同的操作中选择其中的一个，如图5-5、图5-6、图5-7所示。

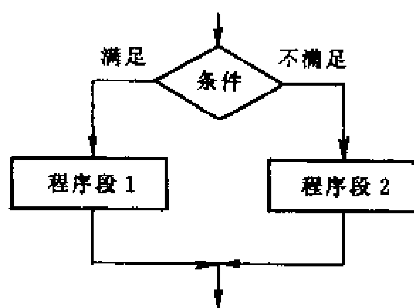


图5-5　选择分支图

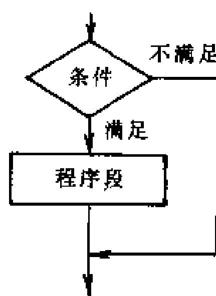


图5-6　简单分支图

在图5-6所示的两个路径中，有一个是不执行任何操作的。图5-7是多分支结构，它是在许多不同的操作中选择其中的一个，究竟选定哪一个操作是由测试表达式来决定的。图 5-6和图5-7只不过是图5-5的变态而已。

3. 循环结构

循环结构是重复做一系列的操作，直到某个条件出现为止。如图5-8和图5-9所示。

图5-8是一种重复型结构，它表示如果某一条件一直成立，则重复做同一个操作或一系列操作，直到条件不成立时为止。它是先检查条件，再去执行操作。图5-9 这种循环结构，先执行

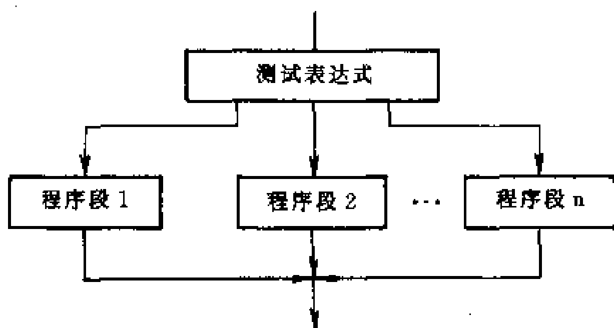


图5-7 情况分支图

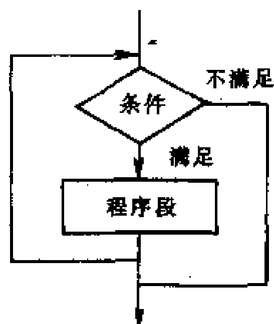


图5-8 WHILE-DO 型循环图

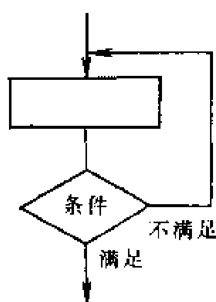


图5-9 REPEAT-UNTIL 型循环图

操作，再去检查条件成立与否，因此，这种结构至少要执行这些操作一次。图5-9 循环结构是由图5-8的循环结构演变而来的，换句话说，任何用 REPEAT-UNTIL 表示的问题都可以用 WHILE-DO 来表示。

（二）一个大型的复杂程序应按其功能分解成若干个功能模块，并把这些模块按层次关系进行组装。每个模块内的执行步骤都应当很清楚，整个程序的算法和每个模块算法，都用标准的结构来表示。每个结构内还可以包含有和自己同样形式的结构或是其他形式的结构，而且每个结构只有一个入口点和一个出口点，以结构化的方式编写的程序容易理解，也容易除错。

（三）在程序设计时，应采用"自上而下、逐步求精"的实施方法。

此方法又称为系统化程序设计方法。在这个方法中，首先把一个大型程序分解成几个主要的模块。最高层次部分说明这些模块之间的关系以及它们的功能，也就是说，最高层次部分是对整个程序的概述。而每个主要的模块再分解成几个较小的模块，然后继续分解成更小的模块，直到每个模块内的操作步骤都很清楚、很容易理解为止，最后把一个模块或几个模块分配给每个程序设计师编写。

与"自上而下"的程序设计方法相对应，还有"自下而上"的程序设计方法，在这个方法中，每个程序设计师开始编写低层次的模块并且期望这些模块最后能组合在一起。如果组合完成了，那么和"自上而下"的设计方法产生相同结果。现在许多程序设计都是混合使用这两种方法，由上而下开始设计，然后由最小的模块开始编写，测试连接，再一直往上做，直到最终完成为止。

按结构化程序设计方法编写出的程序，具有风格优美、结构优良、层次清晰、容易阅读和理解、容易修改和验证等优点。

程序结构化的首要问题是程序的模块化。一个大型程序划分成若干个功能模块，其中有一个模块称为主模块，由它选择和调用其它各个功能模块，被调用的各个模块称为子模块。这种将一个复杂的大型程序按其功能划分为若干相对独立的模块进行程序设计的方法称为程序的模块化。

在汇编语言程序设计中，程序模块化是通过子程序（或过程）的手段来实现的。

三、简单程序设计

简单程序是一种顺序结构的程序，它的执行自始至终按照语句出现的先后顺序进行。这种程序也叫直线程序。

例5.5 求两个数的平均值。这两个数分别放在 x 单元和 y 单元中，而平均值放在 z 单元中。编制程序如下所示：

```
DATAHE  SEGMENT
    x  DB 8CH
    y  DB 64H
    z  DB ?
DATAHE  ENDS
CODEHE  SEGMENT
  MAIN  PROC FAR
        ASSUME CS:CODEHE,DS:DATAHE
      START:  PUSH  DS
              MOV   AX,0
              PUSH  AX
              MOV   AX,DATAHE      ;初始化数据段
              MOV   DS,AX
              MOV   AL,x           ;第一个数送入 AL
              ADD   AL,y           ;两个数相加,结果→AL
              MOV   AH,00H
              ADC   AH,00H         ;带进位加法
              MOV   BL,02H         ;除数2→BL
              DIV   BL             ;AX 除以 BL 的内容,商→AL,余数→AH
              MOV   z,AL           ;结果送入 z 单元
              RET
  MAIN  ENDP
CODEHE  ENDS
        END  START
```

例5.6 在内存中自 tab 开始的16个单元连续存放着0至15的平方值（平方表），任给一个数 x（0≤x≤15）在 x 单元中，例如13，查表求 x 的平方值，并把结果放入 y 单元中。

根据给出的平方表，分析表的存放规律，可知表的起始地址与数 x 之和，正是 x 的平方值所在单元的地址，由此编制程序如下：

```
DATA SEGMENT
    tab     DB 0,1,4,9,16,25,36,49,64,81,100,121,144,169,196,225
    x       DB 13
    y       DB ?
DATA ENDS
```

```
CODE SEGMENT
        ASSUME  CS:CODE,DS:DATA
    START:  MOV  AX,DATA
            MOV  DS,AX
            LEA  BX,tab
            MOV  AH,0
            MOV  AL,x
            ADD  BX,AX
            MOV  AL,[BX]
            MOV  y,AL
            MOV  AH,4CH
            INT  21H
CODE ENDS
        END  START
```

四、分支程序设计

在很多实际问题中，都是根据不同的情况进行不同的处理。这种思想体现在程序设计中，就是根据不同条件而跳到不同的程序段去执行，这就构成了分支程序。在汇编语言程序设计中，跳跃是通过条件转移指令来实现的。

在分支程序中，不论是两分支结构还是多分支结构，它们都有一个共同特点：运行方向是向前的，在某种确定条件下，只能执行两个或多个分支中的一个分支。

例5.7　给定以下符号函数：

$$y=\begin{cases}1,\ 当\ x>0\\0,\ 当\ x=0\\-1,\ 当\ x<0\end{cases}$$

这是一个简单的分支结构，任意给定 x 值，假定为－25，且存放在 x 单元，函数值 y 存放在 y 单元，则根据 x 的值确定函数 y 的值。

流程图如图5-10所示。

编写程序如下：

```
DATAX  SEGMENT
    x    DB -25
    y    DB ?
DATAX  ENDS
CODEX  SEGMENT
  MAIN  PROC FAR
      ASSUME  CS:CODEX,DS:DATAX
    START:  PUSH  DS
            MOV   AX,0
            PUSH  AX
            MOV   AX,DATAX
```

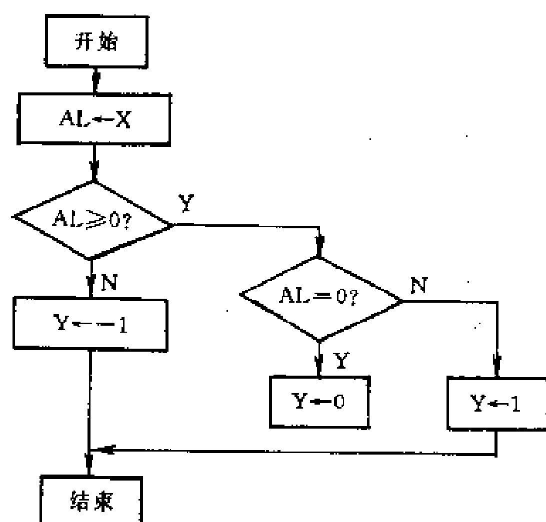


图5-10　实现符号函数程序的流程图

```
                MOV   DS,AX
                MOV   AL,x            ;x→AL
                CMP   AL,0
                JGE   LOOP1
                MOV   AL,0FFH
                MOV   y,AL            ;x<0时,-1送入y单元
                RET
      LOOP1:    JE   LOOP2
                MOV   AL,1
                MOV   y,AL            ;x>0时,1送入y单元
                RET
      LOOP2:    MOV   AL,0
                MOV   y,AL            ;x=0时,0送入y单元
                RET
    MAIN   ENDP
  CODEX   ENDS
      END   START
```

例5.8　设有首地址为arry的字数组,已按升序排好,数组长度为n=15,且数据段与附加段占同一段,在该数组中查找数number,若找到它,则从数组中删掉;若找不到,则把它插入正确位置,且变化后的数组长度在DX中。

根据题意程序编写如下:

```
  DATAJ SEGMENT
           DW ?
    n      DW 15
    number DW 83
    arry   DW 5,10,17,21,28,32,41,50,56,67,72,88,95,125,150
```

```
DATAJ  ENDS
CODMA  SEGMENT
  MAIN  PROC FAR
    ASSUME  CS:CODMA,DS:DATAJ,ES:DATAJ
   START:  PUSH DS
           SUB  AX,AX
           PUSH  AX
           PUSH  ES
           MOV  AX,DATAJ
           MOV  DS,AX
           PUSH DS
           POP  ES
           MOV  AX,number          ;待查找的数放入 AX
           MOV  DX,n               ;初始化 DX
           MOV  CX,n               ;设置计数器 CX
           MOV  DI,OFFSET arry     ;arry 的有效地址放入 DI
           CLD                     ;建立方向标志
           REPNE  SCASW            ;用重复串扫描指令进行查找
           JE  DELETE
           DEC  DX
           MOV  SI,DX
           ADD  SI,DX
   TT3:    CMP  AX,arry[SI]
           JL  TT1
           MOV  arry[SI+2],AX      ;此段程序功能是:若没有查到,
           JMP  TT2                ;则将此数插入正确位置
   TT1:    MOV  BX,arry[SI]
           MOV  arry[SI+2],BX
           SUB  SI,2
           JMP  TT3
   TT2:    ADD  DX,2               ;修改数组长度
           JMP  FAN
   DELETE: JCXZ  NEXT
   LOOPT:  MOV  BX,[DI]            ;此程序段功能是:若查找到,
           MOV  [DI-2],BX          ;则从数组中删除该数
           ADD  DI,2
           LOOP  LOOPT
   NEXT:   DEC  DX                 ;修改数组长度
   FAN:    POP  ES
           RET
```

```
    MAIN   ENDP
CODMA   ENDS
     END   START
```

五、循环程序设计

循环结构的基本特点，我们在本节第二部分已经说明。循环结构和分支结构一样，是应用极为广泛的基本程序结构之一。

(一)循环程序分四部分：

1. 设置循环的初值。如设置循环次数的计数器，为使循环体正常工作而建立的初始状态等。

2. 循环体。循环体是循环工作的主体部分，是为完成某种特定功能而设计的程序段。

3. 修改部分。为保证每次循环时，相关信息(如计数器的值、操作数地址等)能发生有规律的变化，为下次循环作好准备。

4. 循环控制部分。循环控制是循环程序设计的关键。每个循环程序必须选择一个恰当的循环控制条件来控制循环的运行和结束。如果循环不能工作运行，则不能完成特定功能；如果循环不能结束，则将陷入“死循环”。因此，合理地选择循环条件就成为循环程序设计的关键问题。有时循环次数是已知的，可使用循环次数计数器来控制；有时循环次数是未知的，则应根据具体情况设置控制循环结束的条件。

(二)循环程序设计

控制循环是循环程序设计的关键问题。控制循环的方法很多，常用的有：

1. 用计数器控制(循环次数已知)；

2. 按条件控制(循环次数未知)；

3. 用开关变量控制(分支规律已知，计数次数或循环条件已知)；

4. 用逻辑尺控制。

例5.9　从 xx 单元开始的30个连续单元中存放有30个无符号数，从中找出最大者送入 yy 单元中。

根据题意，我们把第一个数先送入 AL 寄存器，将 AL 中的数与后面的29个数逐个进行比较，如果 AL 中的数较小，则两数交换位置；如果 AL 中的数大于等于相比较的数，则两数不交换位置，在比较过程中，AL 中始终保持较大的数，比较29次，则最大者必在 AL 中，最后把 AL 中的数(最大者)送入 yy 单元。

这个问题的特点是循环比较的次数是已知的，因此可以用计数器控制循环，程序的流程图如图5-11所示。

程序编写如下：

```
DATASP SEGMENT
  xx   DB 73,59,61,45,81,107,37,25,14,64
       DB 3,17,9,23,55,97,115,78,121,67
       DB 215,137,99,241,36,58,87,100,74,62
  yy   DB ?
DATASP ENDS
```

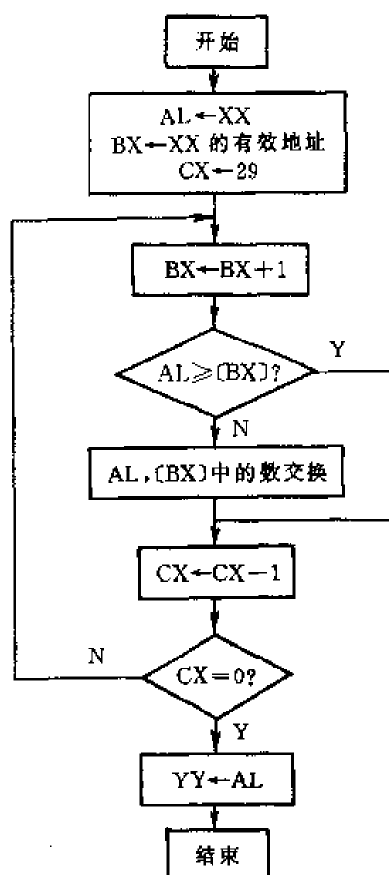


图5-11 从一批数中求最大者流程图

```
CODESP SEGMENT
    ASSUME  CS:CODESP,DS:DATASP
  MAIN  PROC FAR
   START:  PUSH  DS
           MOV   AX,0
           PUSH  AX
           MOV   AX,DATASP
           MOV   DS,AX
           MOV   AL,xx
           MOV   BX,OFFSET xx
           MOV   CX,29
   LOOP1:  INC   BX
           CMP   AL,[BX]
           JAE   LOOP2
           XCHG  AL,[BX]
   LOOP2:  DEC   CX
           JNZ   LOOP1
           MOV   yy,AL
           RET
```

```
    MAIN   ENDP
CODESP   ENDS
      END   START
```

例5.10　从自然数1开始累加,直到累加和大于1000为止,统计被累加的自然数的个数,并把统计的个数送入 n 单元,把累加和送入 sum 单元。

根据题意,被累加的自然数的个数事先是未知的,也就是说,循环的次数是未知的,因此不能用计数器方法控制循环。但题目中给定一个重要条件,即累加和大于1000则停止累加,因此,可以根据这一条件控制循环。我们用 CX 寄存器统计自然数的个数,用 AX 寄存器存放累加和,用 BX 寄存器存放每次取得的自然数。程序的流程图如图5-12所示。

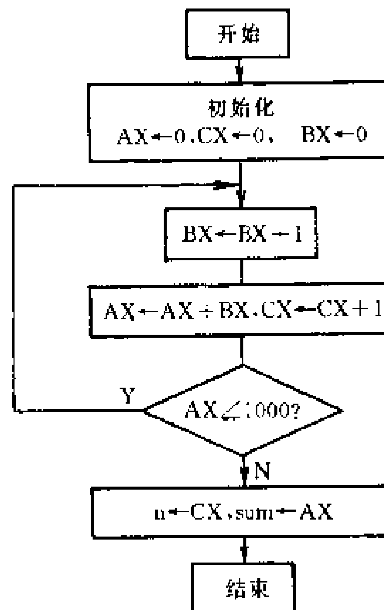


图5-12　利用条件控制循环流程图

程序编写如下:

```
DATAS SEGMENT
    n    DW ?
    sum DW ?
DATAS ENDS
STACK SEGMENT PARA STACK 'stack'
      DW 200 DUP(?)
STACK ENDS
CODES SEGMENT
    MAIN PROC FAR
        ASSUME CX:CODES,DS:DATAS,SS:STACK
      START: PUSH   DS
             MOV    AX,0
             PUSH   AX
             MOV    AX,DATAS
```

```
            MOV   DS,AX
            MOV   CX,0
            MOV   AX,0
            MOV   BX,0
    LOOPT:  INC   BX
            ADD   AX,BX
            INC   CX
            CMP   AX,1000
            JLE   LOOPT
            MOV   n,CX
            MOV   sum,AX
            RET
    MAIN ENDP
CODES ENDS
    END START
```

例5.11　设有两个函数

$y=x+5$

$f=t-27$

根据某种控制过程的需要，函数 y 要计算5次，函数 f 要计算3次，且计算的次序是：y 计算2次，f 计算1次，y 计算1次，f 计算1次，y 计算1次，f 计算1次，最后 y 再计算1次。x 的初值为15，每计算 y 一次，自变量 x 增值2；t 的初值为375，每计算一次 f，其值下降27。第一个函数计算出的5个函数值依次存放在由 y 单元开始的连续5个单元中；第二个函数计算出的3个函数值依次存放在由 f 单元开始的连续3个字单元中。试设计程序。

根据题意，由计算 y 和 f 的顺序，我们可以设计一个逻辑尺：11010101，将其存放在 DL 中，其中1表示计算 y，0表示计算 f。然后利用算术左移指令 SAL，根据有无进位，决定是计算 y 还是计算 f，若每次左移1位，则移位次数共8次。

程序的流程图如图5-13所示。

程序编写如下：

```
DATAS SEGMENT
   x   DB 15
   y   DB 5 DUP(?)
   t   DW 375
   f   DW 3 DUP(?)
DATAS ENDS
STACK SEGMENT PARA STACK 'stack'
      DW 200 DUP(?)
STACK ENDS
CODES SEGMENT
      ASSUME CS:CODES,DS:DATAS,SS:STACK
```

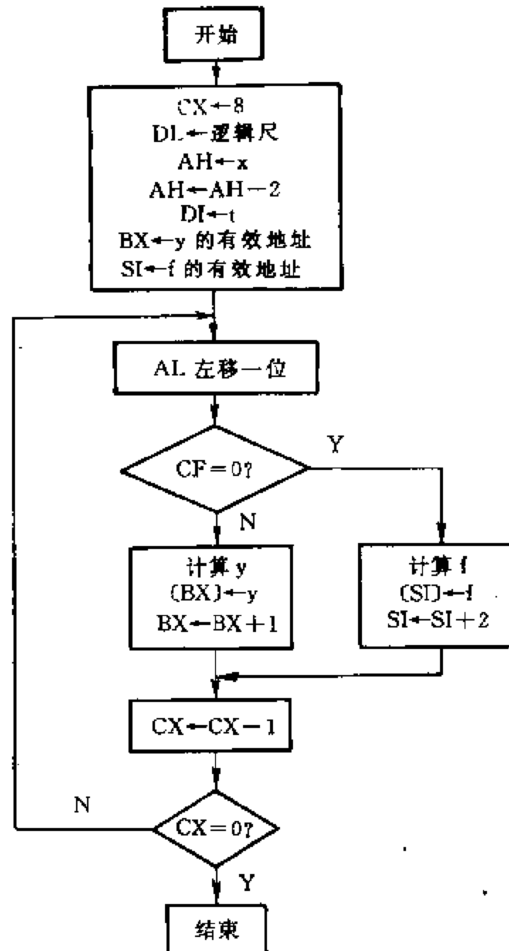


图5-13　用逻辑尺控制循环流程图

```
MAIN PROC FAR
 START:  PUSH   DS
         MOV    AX,0
         PUSH   AX
         MOV    AX,DATAS
         MOV    DS,AX
         MOV    CX,8
         MOV    DL,11010101B
         MOV    AH,x
         SUB    AH,2
         MOV    DI,t
         LEA    BX,y
         LEA    SI,f
 LOOPT:  SAL    DL,1
         JNC    LOOP1
         ADD    AH,2
         MOV    AL,AH
```

```
            ADD   AL,5
            MOV   [BX],AL
            INC   BX
            JMP   LOOP2
    LOOP1:  SUB   SI,27
            MOV   [SI],DI
            ADD   SI,2
    LOOP2:  LOOP  LOOPT
            RET
  MAIN ENDP
CODES ENDS
    END START
```

（三）多重循环程序设计

多重循环又称循环嵌套，即循环套循环。有些问题比较复杂，单重循环难以解决，必须使用多重循环。

在使用多重循环时，必须注意以下几点：

1. 内循环必须完整地包含在外循环内，内外循环不能相互交叉。

2. 内循环在外循环中位置可根据需要任意设置，在分析程序流程时要避免出现混乱。

3. 内循环既可以嵌套在外循环中，也可以几个内循环并列存在。可以从内循环中直接跳到外循环，但不能从外循环直接跳进内循环中

4. 防止出现"死循环"。无论是外循环，还是内循环，千万不要使循环返回到初始部分，否则会出现"死循环"，这一点应当特别注意。

5. 每次通过外循环再次进入内循环时，初始条件必须重新设置。

例5.12　设有4个学生参加5门课程的考试，试计算每个学生的平均成绩和每门课的平均成绩。

根据题意，我们把4名学生的成绩依次存放在一个字数组（首址为 chenggi）中，把每个学生的平均成绩和每门课的平均成绩保存在两个字数组（首址分别为 aver 和 mean）中，流程图如图5-14所示。

编写程序如下：

```
DATARY SEGMENT
   chenggi DW 85,92,76,88,90,
           DW 72,81,68,84,78
           DW 90,86,94,100,80
           DW 75,62,80,79,58
   aver    DW 4 DUP(?)
   mean    DW 5 DUP(?)
DATARY ENDS
CODERY SEGMENT
      ASSUME  CS:CODERY,DS:DATARY
```

```
MAIN PROC FAR
 START: PUSH   DS
        MOV    AX,0
        PUSH   AX
        MOV    AX,DATARY
        MOV    DS,AX
        MOV    CX,4
        LEA    BX,chenggi
        LEA    SI,aver
        SUB    BX,2
 LOOP1: PUSH CX
        MOV    AX,0
        MOV    CX,5
 LOOP2: ADD    BX,2
        ADD    AX,[BX]
        LOOP   LOOP2
        MOV    DL,5
        DIV    DL
        MOV    [SI],AL
        ADD    SI,2
        POP    CX
        LOOP   LOOP1
        MOV    BX,OFFSET chenggi
        MOV    SI,OFFSET mean
        MOV    DI,BX
        MOV    CX,5
 LOOP3: MOV    BX,DI
        SUB    BX,10
        PUSH   CX
        MOV    AX,0
        MOV    CX,4
 LOOP4: ADD    BX,10
        ADD    AX,[BX]
        LOOP   LOOP4
        MOV    DL,4
        DIV    DL
        MOV    [SI],AL
        ADD    SI,2
        ADD    DI,2
        POP    CX
```

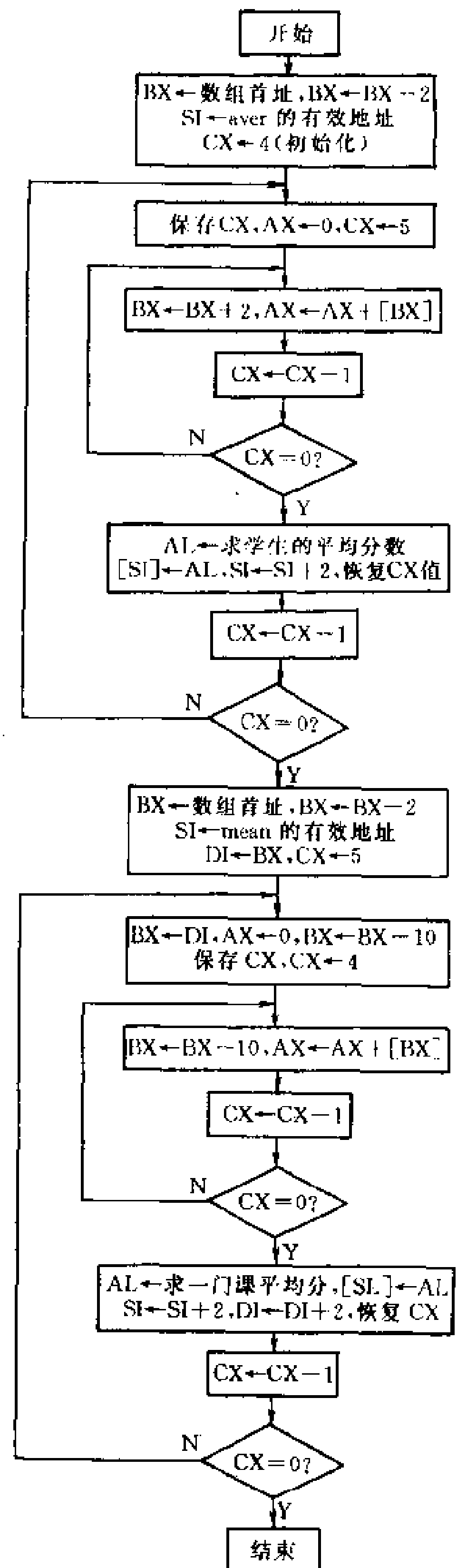


图5-14 计算每个学生和每门课平均成绩流程图

```
            LOOP   LOOP3
            RET
     MAIN ENDP
CODERY ENDS
       END START
```

六、子程序设计

(一)子程序概念

如果在一个程序中的多处需要用到同一段程序,或者说在一个程序中需要多次执行某一连串的指令时,那么我们可以把这段要执行或这一连串的指令抽取出来,写成一个相对独立的程序段,每当我们想要执行这段程序或这一连串的指令时,就调用这段程序,执行完这段程序后再返回原来调用它的程序。这样我们每次执行这段程序时,就不必重写这一连串的指令了,这样的程序段称为子程序或过程。而调用子程序的程序称为主程序或调用程序。

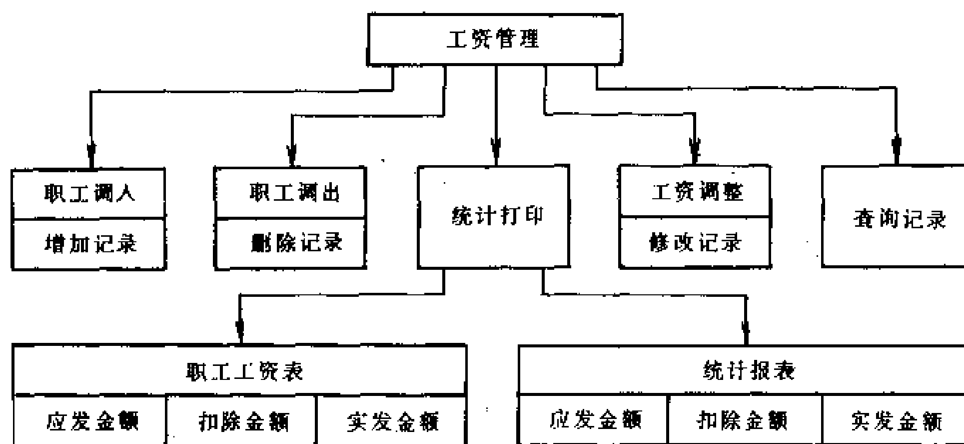


图5-15　工资管理示意图

此外,使用子程序还有另一个主要理由。我们在本节一开始曾讲到"由上而下"的程序设计方法,在这个方法中,我们把整个问题划分成好几个模块,把每个模块再划分成更小的模块,直到每个模块的算法都描述的很清楚为止。图5-15说明在工资管理中,如何以图形方式表示这些模块的层次关系,这样的图表我们通常称之为层次图。这个方法的重点在于它能把一个大的问题划分成许多小的问题,而这些小的问题就构成了一个个相对独立的模块,它们可以单独编程、调试和纠错。这些独立的模块通常编写成子程序,而且可以被层次图之最高层的主程序调用。子程序结构是模块化程序设计的重要工具和手段。

(二)子程序的定义

子程序是用过程定义伪指令 PROC 和 ENDP 来定义的。有关伪指令 PROC 和 ENDP 我们已在前面介绍过了,这里只对其类型属性作一些说明,因为它是一个过程能否正确执行的保证之一。

过程类型属性的确定原则:

1. 调用程序和过程若在同一代码段中,则使用 NEAR 属性;

2. 调用程序和过程若不在同一代码段中,则使用 FAR 属性;

3. 主程序应定义为 FAR 属性。因为我们把程序的主过程看作 DOS 调用的一个子程序，而 DOS 对主过程的调用和返回都是 FAR 属性。

另外，过程定义允许嵌套，即在一个过程定义中允许包含多个过程定义。

例5.13　调用程序和子程序在同一代码段中。

```
CODE   SEGMENT
          ⋮
   MAIN   PROC   FAR
             ⋮
        CALL   PPP1
             ⋮
        RET
     PPP1   PROC   NEAR
               ⋮
            CALL   PPP2
               ⋮
            RET
        PPP2   PROC NEAR
                 ⋮
             RET
        PPP2   ENDP
     PPP1   ENDP
   MAIN   ENDP
CODE   ENDS
```

在本例中过程定义相当于三层嵌套，即主过程 MAIN 嵌套子过程 PPP1，而子过程 PPP1 又嵌套子过程 PPP2。因为主过程和两个子过程均在同一代码段，因此，除主过程使用 FAR 属性外，其它两个子过程均使用 NEAR 属性。

例5.14　调用程序和子程序不在同一代码段。

```
CODE1   SEGMENT
           ⋮
   RRR   PROC   FAR
            ⋮
       RET
   RRR   ENDP
CODE1   ENDS
CODE2   SEGMENT
           ⋮
     CALL RRR
           ⋮
CODE2   ENDS
```

本例中，因为子程序 RRR 和调用程序不在同一代码段，因此子程序 RRR 应定义为 FAR 属性，这样调用指令 CALL 和返回指令 RET 都是 FAR 属性的。

应当指出，在一个过程中可以有一个以上的 RET 指令，这完全是根据需要而设置的。

(三)子程序的调用和返回

子程序的调用和返回由 CALL 和 RET 指令完成，子程序的正确调用和正确返回是正确执行子程序的保证。

为了使子程序正确地执行，有两点应特别注意：

1. 正确选择过程的属性。

2. 正确使用堆栈，因为在调用程序中执行 CALL 指令时将把断点地址压入堆栈。这个地址正是由子程序返回到调用程序的地址。当在子程序中执行 RET 指令时，便把这个返回地址由堆栈弹出(称为恢复断点)，返回调用程序自此继续往下执行。若在子程序中不能正确地使用堆栈，而造成执行 RET 前堆栈指针 SP 并未指向进入子程序时的返回地址，则必然会导致运行出错。因此在子程序中使用堆栈要特别注意。

(四)寄存器的保护与恢复

在程序设计中，调用程序(或主程序)与子程序通常是独立编写的，因此它们所使用的一些寄存器和存储单元经常会发生冲突。如果调用程序在调用子程序以前的某些寄存器或存储单元的内容在从子程序返回到调用程序后还要使用，而子程序又恰好使用了这些寄存器或存储单元，则这些寄存器或存储单元的原有内容遭到了破坏，那就会使程序运行出错，为防止这种错误的发生，在进入子程序之前或之后应该把子程序所使用的寄存器或存储单元的内容保存在堆栈中，而退出子程序之前再恢复原有的内容。在主、子程序间传送参数的寄存器不需要保护。

寄存器的保护有两种方法：

1. 把需要保护和恢复的寄存器的内容，在调用程序中压入堆栈和弹出堆栈。这种方法有一个好处，就是在每次调用子程序时，只要把你所关心的寄存器压入堆栈，返回后弹出即可。但缺点是，在调用程序中使用压入和弹出堆栈的功能会使调用程序不容易理解，而且可能在调用程序其它地方使用某个寄存器时，却忘了把它压入堆栈内。

2. 进入子程序后首先把需要保护的寄存器的内容压入堆栈，而在返回调用程序前再恢复这些寄存器的内容。这种方法是我们所喜欢的。这种方法的好处是：首先，我们在调用程序中的任何地方都可调用子程序，而不会破坏任何寄存器的原有内容。其次，这种方法只需要写一次压入和弹出堆栈群即可。例如：

```
DUBT   PROC NEAR
       PUSH  AX
       PUSH  BX
       PUSH  CX
       PUSH  DX
        ⋮
       POP  DX
       POP  CX
       POP  BX
       POP  AX
```

```
    RET
DUBT ENDP
```

注意:堆栈的工作方式是后进先出。

(五)调用程序与子程序之间的参数传递

调用程序在调用子程序时,往往需要向子程序传递一些参数;同样的子程序运行后也经常要把一些结果参数传回给调用程序。调用程序与子程序之间的这种信息传递称为参数传递。

参数传递有三种主要的方式:

1.通过寄存器传递参数

这种方式适合于传递参数较小的一些简单程序。

例5.15 把一个2位压缩的BCD码转换成二进制数。

我们先把这个BCD数放在寄存器AL中,然后由主程序传递给子程序BCD_BINARY,子程序又把这个BCD数的二进制值放在AL内传递回调用程序,并保存在VALUE单元中,在子程序一开始,我们把程序状态字寄存器及其他在程序中会用到的寄存器内容压入堆栈内,但AX寄存器不需要压入和弹出堆栈,因为我们是用AX把一个值传递到子程序中,同时子程序把结果值放在AX中,而传回给主程序的。程序编写如下:

```
DATA_BIN  SEGMENT
  BCD_IN   DB ?                ;存放BCD值
  VALUE   DB ?                 ;存放二进制值
DATA_BIN  ENDS
CODE  SEGMENT
    ASSUME  CS:CODE,DS:DATA_BIN
  MAIN  PROC FAR
      PUSH  DS
      MOV  AX,0
      PUSH  AX
      MOV  AX,DATA_BIN
      MOV  DS,AX
      MOV  AL,BCD_IN           ;把待转换的BCD码送AL寄存器
      CALL  BCD_BINARY
      MOV  VALUE,AL
      RET
  MAIN  ENDP
  BCD_BINARY  PROC NEAR
      PUSHF                    ;保存程序状态字
      PUSH  BX                 ;保存BX和CX
      PUSH  CX
      MOV  AH,AL               ;把BCD数送入AH
      AND  AH,0FH              ;分离和保存BCD数的低位数字
      MOV  BL,AH
```

```
        AND   AL,0F0H              ;分离 BCD 数的高位数字
        MOV   AL,04
        ROR   AL,CL                ;把 BCD 数高位数字移到低位
        MOV   BH,0AH               ;把转换因子送入 BH
        MUL   BH                   ;AL 中 BCD 数的高位数字乘以 BH 中的0AH
                                   ;结果在 AX 中
        ADD   AL,BL                ;把相乘结果与 BCD 码的低位数字相加最终
                                   ;结果送入 AL 中
        POP   CX
        POP   BX
        POPF                       ;恢复被保护的寄存器
        RET
    BCD_BINARY   ENDP
CODE   ENDS
      END MAIN
```

2. 通过地址表传递参数地址

这种方式适合于参数较多的情况，但要求事先建立一个地址表，通过地址表传递参数的地址，地址表可以在内存中或外设端口中。

例5.16　对例5.15我们通过传送参数地址的方法重新编写程序如下：

```
DATA   SEGMENT
  BCD_IN   DB ?
  VALUE   DB ?
DATA   ENDS
CODE   SEGMENT
     ASSUME   CS:CODE,DS:DATA
    MAIN   PROC FAR
         PUSH   DS
         MOV    AX,0
         PUSH   AX
         MOV    AX,DATA
         MOV    DS,AX
         MOV    SI,OFFSET BCD_in       ;把待转换的 BCD 码有效地址送 SI
         MOV    DI,OFFSET value
         CALL   BCD_BINARY
         MOV    [DI],AL
         RET
    MIAN   EMDP
    BCD_BINARY   PROC NEAR
         PUDHF
```

```
            PUSH  BX
            PUSH  CX
            MOV   AL,[SI]          ;把 BCD_IN 单元中的数送入 AL
            MOV   AH,AL
            AND   AH,0FH
            MOV   BL,AH
            AND   AL,0F0H
            MOV   CL,4
            ROR   AL,CL
            MOV   BH,0AH
            MUL   BH
            ADD   AL,BL
            POP   CX
            POP   BX
            POPF
            RET
      BCD_BINARY  ENDP
CODE  ENDS
      END  MAIN
```

3.通过堆栈传递参数

为了利用堆栈传递参数,必须在主程序中任何调用子程序之前的地方,把这些参数压入堆栈,然后利用在子程序中的指令从堆栈弹出而取得参数。同样,要从子程序传递回调用程序的参数也被压入堆栈内,然后由主程序中的指令把这些参数从堆栈中取出。下面我们举例说明如何利用堆栈来传递参数。

例5.17　试将一个2位十进制数的压缩的 BCD 码转换成十六进制数，并在屏幕上显示出来。

根据题目的要求,我们利用堆栈传递参数的方法。首先在主程序中把 BCD 码压入堆栈,在子程序 BCD_BIN 中从堆栈中取出这个 BCD 码,然后先把它转换成二进制数,并将其保存在堆栈。返回主程序后再将这个二进制数从堆栈中取出,再把它转换成十六进制数的 ASCII 码,最后把这个十六进制数的 ASCII 码在屏幕上显示出来。程序编写如下:

```
DATAH  SEGMENT
   BCDMA   DB ?
DATAH ENDS
STACK   SEGMENT PARA STACK 'STACK'
           DW 100 DUP(?)
   TOS     LABLE WORD
STACK   ENDS
CODEH   SEGMENT
     ASSUME  CS:CODEH,DS:DATAH,SS:STACK
```

```
MAIN    PROC FAR
 START:  MOV   AX,STACK
         MOV   SS,AX
         MOV   SP,OFFSET TOS
         PUSH  DS
         MOV   AX,0
         PUSH  AX
         MOV   AX,DATAH
         MOV   DS,AX
         MOV   AH,00H
         MOV   AL,BCDMA
         PUSH  AX              ;把 BCD 码压入堆栈
         CALL  BCD_BIN         ;调用子程序把 BCD 码转化成二进制数
         POP   AX              ;把转化后的二进制数弹出送入 AX
         MOV   CH,2
 LOOP1:  MOV   CL,4
         ROL   AL,CL
         MOV   BL,AL
         AND   BL,0FH
         ADD   BL,30H
         CMP   BL,3AH
         JL    LOOP2
         ADD   BL,07H
 LOOP2:  MOV   DL,BL
         MOV   AH,02H
         INT   21H
         DEC   CH
         JNZ   LOOP1
         RET
MAIN    ENDP
BCD_BIN   PROC NEAR
        PUSH  AX
        PUSHF
        PUSH  BX
        PUSH  CX
        PUSH  BP
        MOV   BP,SP
        MOV   AX,[BP+12]
        MOV   AH,AL
        AND   AH,0FH
```

```
        MOV   BL,AH
        AND   AL,0F0H
        MOV   CL,4
        ROR   AL,CL
        MOV   BH,0AH
        MUL   BH
        ADD   AL,BL
        MOV   [BP+12],AX
        POP   BP
        POP   CX
        POP   BX
        POPF
        POP   AX
        RET
    BCD_BIN  ENDP
  CODEH   ENDS
      END   MAIN
```

为了说明本例利用堆栈传递参数的过程，请参看图5-16所示的堆栈图。

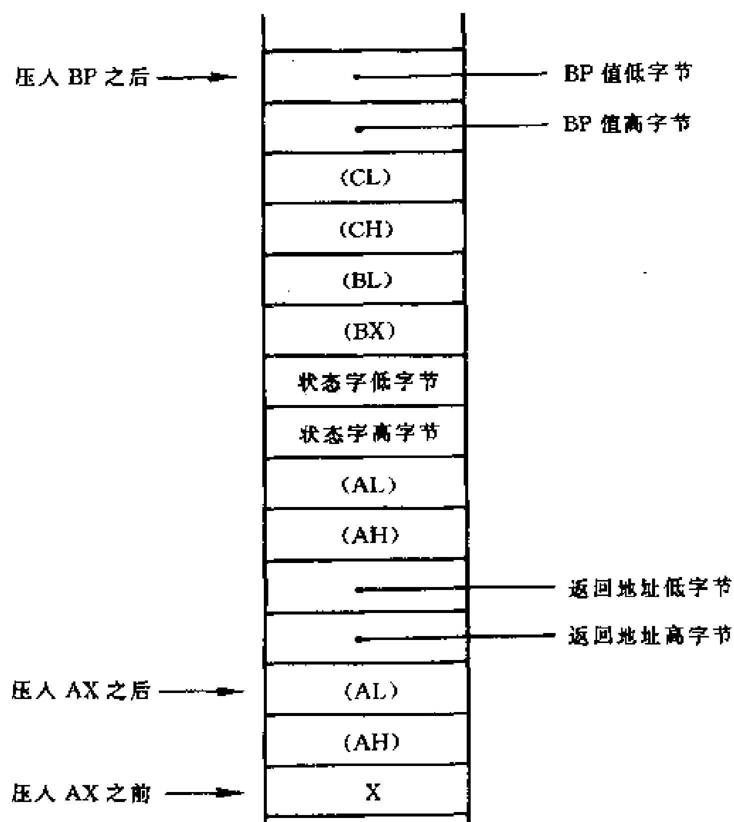


图5-16　例5.17的堆栈示意图

利用堆栈传递参数有两个非常重要的问题：

(1)当使用堆栈来传递参数时，有一个应当注意的潜在问题就是堆栈溢出(stack-

overflow)。每当你用堆栈传送参数时，你应当非常清楚，已经把什么东西压入了堆栈内及在子程序中每个地方的堆栈指针是指向哪里，弄不好，会引起混乱和造成堆栈溢出。所谓堆栈溢出是指堆栈超出了为它开辟的存储空间。

(2)8086/8088有四种形式的 RET 指令，一般的近程 RET 指令能把返回地址由堆栈弹入到 IP，同时把堆栈指针加2；一般的远程 RET 指令能由堆栈把返回的 IP 及 CS 值弹入到 IP 及 CS，同时把堆栈指针加4，其他二种 RET 指令形式分别执行相同的功能。但是它们把一个在指令中指定的数字加入堆栈指针。如近程 RET 6指令会从堆栈弹出一个字的内容到 IP 同时把堆栈指针加2，然后再加6到堆栈指针，这是一种快速方式，可以让堆栈指针往下(地址增大方向)跳过一些参数。

(六)子程序的嵌套

一个子程序可以作为调用程序去调用别的子程序，这种结构称为子程序的嵌套。只要有足够的堆栈空间，嵌套的层次是不限的，其嵌套的层数称为嵌套深度。

当调用程序去调用子程序时，将产生断点，而子程序执行完后返回到调用程序的断点处，使调用程序继续往下执行。因此，对于嵌套结构，断点的个数等于嵌套的深度。如图5-17所示。

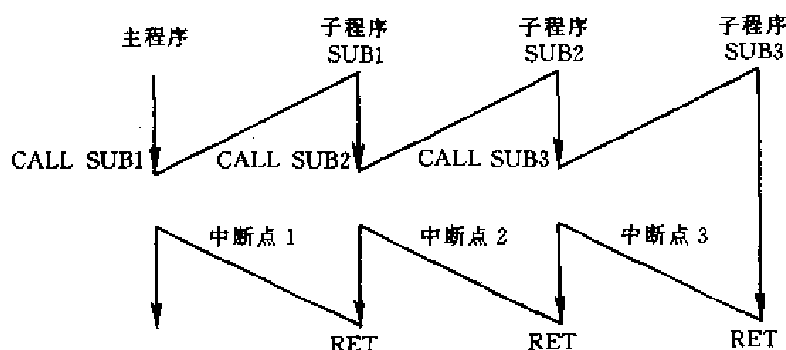


图5-17 子程序嵌套示意图

图5-17所示的嵌套结构其嵌套深度为3，所以断点个数也为3，由此可见，嵌套结构是一种层次结构。

例5.18 设有两个无符号数125和378，其首地址为 x，求它们的和，将结果存放在 SUM 单元。并将其和转换为十六进制数且在屏幕上显示出来。

根据题意，我们设计一个主程序 MAIN 和两个子程序 PROCEDP 和 PROCEDX。在主程序中完成必要的初始化，然后调用子程序。在子程序 PROCEDP 中完成两个数的求和，并把求和结果送入指定单元。然后子程序 PROCEDP 去调用子程序 PROCEDX，把和数转化为十六进制数并在屏幕上显示出来。显然这是一个嵌套结构，其嵌套深度为2。程序编写如下：

```
DATAP  SEGMENT
  x     DW 125,378
  sum  DW ?
DATAP  ENDS
CODEP  SEGMENT
   MAIN  PROC FAR
        ASSUME  CS:CODEP,DS:DATAP
```

```
    START: PUSH DS
           XOR   AX,AX
           PUSH  AX
           MOV   AX,DATAP
           MOV   DS,AX
           MOV   SI,OFFSET x       ;把 x 的有效地址送入 SI
           CALL  PROCDP
           RET
MAIN  ENDP
PROCDP  PROC NEAR
           PUSH  DX                ;寄存器保护
           PUSH  BX
           PUSH  AX
           PUSH  SI
           PUSH  CX
           MOV   AX,[SI]           ;把第一个数送入 AX
           ADD   AX,[SI+2]         ;两个数相加
           MOV   sum,AX            ;把和送入指定单元
           CALL  PROCDX
           POP   CX                ;寄存器恢复
           POP   SI
           POP   AX
           POP   BX
           POP   DX
           RET
PROCDP  ENDP
PROCDX     PROC NEAR
           MOV   BX,sum
           MOV   CH,4
    T1:    MOV   CL,4
           ROL   BX,CL             ;循环左移4次
           MOV   AL,BL             ;屏蔽高4位
           AND   AL,0FH
           ADD   AL,30H            ;化为 ASCII 码
           CMP   AL,3AH            ;ASCII 码与3AH 比较
           JL    T2
           ADD   AL,07H            ;ASCII 码在 A~F
    T2:    MOV   DL,AL             ;ASCII 码在0~9
           MOV   AH,2
```

```
            INT   21H              ;DOS 系统功能调用,显示一个字符
            DEC   CH
            JNZ   T1
            RET
    PROCDX  ENDP
CODEP   ENDS
        END   MAIN
```

七、DOS 系统功能调用。

(一)概述

在汇编语言程序设计中,经常要用到 ROM-BIOS 的一些软中断和系统功能调用来扩充汇编语言的功能。

1. ROM-BIOS(基本 I/O 系统)是固化在 ROM 中的一组 I/O 设备驱动程序,它为系统各主要部件提供设备级的控制,还为汇编语言程序设计者提供了字符 I/O 操作。程序员在使用 ROM-BIOS 的功能模块时,可以不关心硬件 I/O 接口的特性,仅使用指令系统的软中断指令(INT n),这称为中断调用。

例如,将一个 ASCII 字符显示在屏幕的当前所在位置,可使用 ROM-BIOS 的中断类型号 10H,功能号为0EH。程序段如下:

```
MOV   AL,"?"      ;要显示的字符送入 AL
MOV   AH,OEH      ;功能号送入 AH
INT   10H         ;调用10H 软中断
```

2. 系统功能调用是微机的磁盘操作系统 DOS 为用户提供的一组例行子程序,因而又称为 DOS 系统功能的调用。这些子程序可分为以下三个主要方面:

(1)磁盘的读/写及控制管理;

(2)内存管理;

(3)基本输入/输出管理(如键盘、打印机、显示器、磁带管理等),另外还有时间、日期等子程序。

(二)系统功能调用方法

为了使用方便,已将所有子程序顺序编号。例如,基本输入/输出管理中的功能调用1号(键盘输入)、2号(显示字符)、5号(打印字符)、9号(显示字符串)及0A 号(接收键盘输入的字符串)等。

对于所有的功能调用,使用时一般需要经过以下三个步骤:

(1) 子程序的入口参数送相应的寄存器;

(2) 子程序编号送 AH;

(3) 发出中断请求:INT 21H(系统功能调用指令)

例如,显示一个字符串:"Good morning!"

```
MSG   DB 'Good morning! $'
        ⋮
      MOV   DX,OFFSET MSG
```

```
MOV   AH,9
INT   21H
```

有的子程序不需要入口参数,这时(1)可以略去。

例如:

```
MOV   AH,4CH
INT   21H
```

子程序调用结束后,一般都有出口参数,这些出口参数常放在寄存器中,通过出口参数用户可以知道调用的成功与否。

(三)系统功能调用分组说明

系统功能调用可以分组如下(以 DOS3.10版本为例):

0~0CH:传统的字符 I/O 管理。包括键盘、显示器、打印机、异步通讯口的管理。

0D~24H:传统的文件管理。包括复位磁盘,选择磁盘,打开文件,关闭文件,查找目录项,删除文件,顺序读、写文件,建立文件,重新命名文件,查找驱动器分配表信息,随机读、写文件,查看文件长度等。

25~26H:传统的非设备系统调用。包括设置中断向量,建立新程序段。

27~29H:传统的文件管理。包括随机块读写,分析文件名。

2A~2EH:传统的非设备系统调用。读取、设置日期、时间等。

2F~38H:扩充的系统调用组。包括读取 DOS 版本号,中止进程,读取中断向量,读取磁盘空闲空间等。

39~3BH:目录组。包括建立子目录,修改当前目录,删除目录项。

3C~46H:扩充的文件管理组。包括建立、打开、关闭文件,从文件或设备读取数据。在指定的目录里删除、移动、读、写文件,读取、修改文件属性,设备 I/O 控制,复制文件标记等。

47H:目录组。取当前目录。

48~4BH:扩充的内存管理组。包括分配内存,释放已分配的内存,分配内存块,装入或执行程序等。

4C~4FH:扩充的系统管理组。包括中止进程,查询子进程的返回代码,查找第一个相匹配的文件,查找下一个相匹配的文件。

50~53H:扩充的系统调用,DOS 内部使用。

54~62H:扩充的系统调用。包括读取校验状态,重新命名文件,设置读取日期和时间。

从39H 以后的文件管理系统调用是为了处理树形目录结构而提供的。下面我们选择其中一部分常用的系统功能调用分别加以介绍,其余部分可参考有关的 DOS 资料。

(四)基本 I/O 功能调用

1. 带显示键盘输入(1号调用)

1号系统功能调用等待从标准输入设备输入一个字符,并送入寄存器 AL,不需入口参数。例如:

```
MOV   AH,1
INT   21H
```

执行上述指令,系统将扫描键盘,等待有键按下。一旦有键按下,就将键值(相应字符的

ASCII 码值)读入,先检查是否是 Ctrl-Break,若是,则退出命令执行;否则将键值送入 AL 寄存器,同时将这个字符显示在屏幕上。

2. 键盘输入但无显示(8号调用)

8号调用与1号调用类同,只是不在屏幕上显示输入的字符。

3. 打印输出(5号调用)

把 DL 中的字符输出到打印机。例如:

```
MOV  DL,'A'
MOV  AH,5
INT  21H
```

4. 直接控制台输入/输出(6号和调用)

6号调用可以从标准输入设备输入字符,也可以向屏幕上输出字符,并且不检查 Ctrl-Break。

若 DL=FFH 时,表示从键盘输入,若标志 ZF=0,表示 AL 中为键入的字符值;若标志 ZF=1,表示 AL 中不是键入的字符值,即尚无键按下。

若 DL≠FFH 时,表示向屏幕输出,DL 中为输出字符的 ASCII 码值。例如:

```
MOV  DL,0FFH
MOV  AH,6
INT  21H
```

为从键盘输入字符。

```
MOV  DL,24H
MOV  AH,6
INT  21H
```

将24H 对应的字符"$"输出,即从 CRT 屏幕上显示"$"。

5. 直接控制台输入但不显示(7号调用)

等待从标准输入设备输入字符,然后将其送入 AL,如同6号调用,对字符不做检查。

6. 输出字符串(9号调用)

调用时,要求 DS:DX 必须指向内存中一个以"$"作为结束标志的字符串。字符串中每一个字符(不包括结尾标志$)都输出显示或打印。例如:

```
DATA  SEGMENT
  BUF     DB 'HOW DO YOU DO? $'
            ⋮
DATA  ENDS
CODE  SEGMENT
            ⋮
            MOV  AX,DATA
            MOV  DS,AX
            ⋮
            MOV  DX,OFFSET BUF
```

```
        MOV  AH,9
        INT  21H
         ⋮
CODE  ENDS
```

执行本程序,屏幕上将显示:HOW DO YOU DO?

7. 字符串输入(0AH 号调用)

从键盘接收字符串到内存输入缓冲区,要求事先定义一个输入缓冲区,缓冲区内第一个字节指出缓冲区能容纳的字符个数,不能为零。第二个字节保留以用作填写输入的字符实际个数。从第三个字节开始存放从键盘上接收的字符。若实际输入的字符数少于定义的字节数,缓冲区内其余字节填零,若多于定义的字节数,则后来输入的字符丢掉,且响铃。调用时,要求DS:DX 指向输入缓冲区。例如:

```
DATA  SEGMENT
   BUF  DB 50           ;缓冲区长度
        DB ?            ;保留为填入实际输入的字符个数
        DB 50 DUP(?)    ;定义50个字节存储空间
         ⋮
DATA  ENDS
CODE  SEGMENT
         ⋮
        MOV  AX,DATA
        MOV  DS,AX
         ⋮
        MOV  DX,OFFSET BUF
        MOV  AH,10      ;即0AH 送 AH
        INT  21H
         ⋮
CODE  ENDS
```

例5.19 利用 DOS 系统功能调用实现人机对话。

下述程序可以在屏幕上显示一行提示信息,然后接收用户从键盘输入的信息并将其存入内存缓冲区。

```
DATA  SEGMENT
    PARS  DB 100                        ;定义输入缓冲区
          DB ?
          DB 100 DUP(?)
    MESG  DB 'WHAT IS YOUR NAME ?'      ;要显示的提示信息
          DB '$'                        ;提示信息结束标志
DATA  ENDS
STACK  SEGMENT PARA STACK 'STACK'
        DB  100 DUP(?)
```

```
STACK   ENDS
CODE   SEGMENT
    ASSUME   CS:CODE,DS:DATA,SS:STACK
   SART   PROC   FAR
          PUDH   DS
          MOV   AX,0
          PUSH   AX
          MOV   AX,DATA
          MOV   DS,AX
   DISP:  MOV DX,OFFSET MESG
          MOV   AH,9                    ;利用9号功能调用显示提示
          INT   21H
          KEYBD:MOV   DX,OFFSET PARS
          MOV   AH,10                   ;利用10号功能调用接收键盘输入
          INT   21H
          RET
   SART   ENDP
CODE   ENDS
    END   SART
```

第七节　程序设计举例

前面我们已经介绍了程序设计的基本方法。本节将给出一些程序设计例子，这些例子涉及一些常用程序的设计，希望读者通过例子能学习到一些程序设计的技巧和方法。

一、十进制数算术运算

(一)非压缩型 BCD 码的算术运算

1. 多位非压缩型 BCD 码加法

例5.20　内存中从 FIRST 和 SECOND 开始的单元中分别存放着两个4位非压缩型的 BDE 码，数据存放的规则是，低位在低地址，高位在高地址。试编程求这两个数的和，并存放到从 THIRD 开始的内存单元中。

两个4位的十进制数进行加法运算，最高位可能产生进位，为了避免丢失最高位的进位，我们给被加数、加数以及结果存放单元都开辟了5个字节单元。在进行加法运算时，每进行一位加法运算，都将产生的进位加到加数的下一位上，以保证下一位加法操作的正确性。

程序清单

```
STACK   SEGMENT PARA STACK 'STACK'
        DB 128 DUP(0)
STACK   ENDS
DATA   SEGMENT
```

```
    FIRST   DB 4,7,9,2,0               ;十进制数02974
    SECOND DB 6,2,3,7,0                ;十进制数07326
    THIRD   DB 5 DUP(0)
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA,ES:DATA,SS:STACK
    UNPACK  PROC  FAR
        MOV   AX,DATA
        MOV   DS,AX
        MOV   ES,AX
        LEA   SI,FIRST
        LEA   BX,SECOND
        LEA   DI,THIRD
        MOV   CX,4+1                   ;进行5次加法
        CLD
RETRY: LODSB
        ADD   AL,[BX]                  ;对应的一位进行加法运算
        AAA
        STOS                           ;保存当前位和
        INC   BX
        ADC   BYTE PTR[BX],0           ;进位值加到加数的下一位
        LOOP  RETRY
        MOV   AH,4CH
        INT   21H
    UNPACK ENDP
 CODE   ENDS
      END  UNPACK
```

2. 非压缩 BCD 码乘法

例5.21　一个多位的非压缩型 BCD 码与一个一位的非压缩型 BCD 码的乘法应该照如下的方法进行乘法运算：从被乘数中一次取一位（先取低位，后取高位）与乘数相乘并进行十进制乘法调整，得到一个部分积在 AX 中（其中 AH 中是部分积的高位，AL 中是部分积的低位）。调整后的低位要加上前一次的高位，再进行加法调整，然后进行存储，如果加法调整中有进位，则被自动地加到部分积的高位 AH 中，将 AH 中的数存放在结果单元中，以备下一位乘法操作以后作为部分积的高位所使用。

程序清单：

```
STACK   SEGMENT STACK 'STACK'
        DB 128 DUP(0)
STACK   ENDS
DATA    SEGMENT
```

```
    data1   DB 6,5,3,8            ;被乘数8356
    data2   DB 9                  ;乘数9
    data3   DB 5 DUP(0)           ;存放5位积
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA,ES:DATA,SS:STACK
  UNPACK  PROC  FAR
  START:  MOV   AX,DATA
          MOV   DS,AX
          MOV   ES,AX
          MOV   SI,OFFSET data1
          MOV   DI,OFFSET data3
          MOV   CX,4
          CLD
  RETRY:  LODSB
          MUL   data2             ;求部分积
          AAM                     ;十进制乘法调整
          ADD   AL,[DI]           ;加上前一次的高位
          AAA                     ;十进制加法调整
          STOSB                   ;存部分积的低位
          MOV   [DI],AH           ;存部分积的高位
          LOOP  RETRY
          MOV   AH,4CH
          INT   21H
  UNPACK  ENDP
CODE    ENDS
        END  START
```

(二)压缩型BCD码的算术运算

1. 多位压缩型BCD码减法

例5.22 内存中从FIRST和SECOND开始的单元中分别存放着两个3位压缩型BCD码，数据存放的规则是：低位在低地址，高位在高地址。试编程求这两个数的差，并存放到从THIRD开始的内存单元中。要求：如果被减数大于减数，则将“+”号存放在结果的最后，否则，用减数减去被减数，并将“-”存放在结果的最后。程序主要部分为：

```
                  ⋮
first    DB 67H,83H,32H          ;十进制数328367
second   DB 41H,54H,59H          ;十进制数595441
third    DB 0,0,0,'+'
               ⋮
          MOV   SI,OFFSET first
```

```
        MOV   BX,OFFSET second
        MOV   DI,OFFSET third
        MOV   CX,3              ;进行3次减法
        CLC
        CLD
RETRY:  LODSB
        SBB   AL,[BX]           ;对应的一位进行减法运算
        DAS
        STOSB                   ;保存当前位的差
        INC   BX
        LOOP  RETRY
        JC   MANUS              ;被减数小于减数,转 MANUS
        JMP  PROGEND            ;差为正数,转结束
MANUS:  MOV AL,'-'
        STOSB
        MOV   SI,OFFSET second  ;减数当做被减数
        MOV   BX,OFFSET first   ;被减数当做减数
        MOV   DI,OFFSET third
        MOV   CX,3
        CLC
RETRY1: LODSB
        SBB   AL,[BX]
        DAS
        STOSB
        INC   BX
        LOOP  RETRY1
PROGEND:MOV   AH,4CH
        INT   21H
        ⋮
```

二、代码转换

计算机在通信、管理、控制等各种应用中经常会遇到代码转换问题。比如,我们从键盘上输入一个数字,机器中接收到的是它的 ASCII 码值。要让这个数字参加二进制运算,则首先要将其转换成二进制数字,若要参加十进制运算,应先转换成十进制数字。再如内存中的二进制数字要在屏幕上以十进制形式显示出来,则首先要将其转换成十进制数, 再转换成 ASCII 码才能输出。另外对于许多应用实例来说,代码转换是必需的。

常用的代码有二进制、八进制、十进制、ASCII 码、BCD 码、七段显示代码等等。这里将介绍几种主要的代码间的转换。

1. BCD 码转换成二进制数

例5.23 编程将 AX 中的四位 BCD 码转成二进制数,转换的结果存在 AX 中。

假设：AX=9827H，即九千八百二十七。

算法：[(千位数×10+百位数)×10+十位数]×10+个位数，即乘十相加。

编程：(子程序段部分)

```
W10         DW 10           ;十进制数权值
;入口参数   AX=BCD 码
;出口参数   AX=二进制数
AXBCDTO2  PROC NEAR
            PUSH  BX
            PUSH  CX
            PUSH  DX
            MOV   BX,AX     ;保存 AX 中的 BCD 码到 BX
            MOV   AX,0      ;结果单元清零
            MOV   CX,4      ;共处理4位 BCD 码
RETRY:      PUSH  CX
            MOV   CL,4
            ROL   BX,CL     ;一位 BCD 码移到 BX 中的低半字节
            POP   CX
            MUL   W10       ;累加和乘以权值送 AX
            PUSH  BX
            AND   BX,0FH    ;屏蔽 BX 的高半字节
            ADD   AX,BX     ;累加下一位 BCD 码
            POP   BX
            LOOP  RETRY
            POP   DX
            POP   CX
            POP   BX
            RET
AXBCDTO2    ENDP
```

2. 二进制数转换成 BCD 码

例5.24　编程将 AX 中的二进制数转换成四位 BCD 码，转换的结果存在 AX 中。

假设：AX=9827H

算法：把 AX 中的二进制数除以1000得到的商是千位上的 BCD 码，所得余数除以100 得到的商是百位上的 BCD 码，所得余数再除以10得到的商是十位上的 BCD 码，最后所得的余数是个位上的 BCD 码。

程序清单

```
W1000       DW 1000,100,10,1            ;十进制数千，百，十，个位权值
                                        ;入口参数 AX=二进制数
                                        ;出口参数 AX=BCD 码
AX2TOBCD: PROC   NEAR
```

```
            PUSH   SI
            PUSH   BX
            PUSH   CX
            PUSH   DX
            XOR    BX,BX                    ;BCD码暂存单元清零
            MOV    SI,OFFSET W1000          ;权值首地址送SI
            MOV    CX,4                     ;循环次数4送CX
RETRY:      PUSH   CX
            MOV    CL,4
            SHL    BX,CL
            MOV    DX,0
            DIV    WORD PTR[SI]             ;除以权值
            OR     BX,AX                    ;压缩BCD码
            MOV    AX,DX                    ;送余数到AX
            POP    CX
            INC    SI
            INC    SI                       ;地址加2,指向下一个权值
            LOOP   RETRY
            MOV    AX,BX                    ;BCD码由BX送AX
            POP    DX
            POP    CX
            POP    BX
            POP    SI
            RET
AX2TOBCD:   ENDP
```

3. ASCII 码转换成二进制数

例5.25　从 ASCBUF 开始的内存单元中有4个连续的 ASCII 码，要求把它们转换为二进制数放在 BINVAL 单元中。

假设：这4个 ASCII 码为"2683"

算法：把4个 ASCII 码分别转换为0～9的数，然后分别乘以各自的权值，最后累加起来即得到相应的二进制数。

程序清单

```
ASCBUF      DB '6472'                ;ASCII码字符串
ASCLEN      DB $-ASCBUF              ;ASCII码字符个数
BINVAL      DW 0                     ;二进制数结果
MULT10      DW 1                     ;十进制权值存放单元
ASCTOBIN    PROC NEAR
            MOV   CX,10              ;乘法因子送CX
            LEA   SI,ASCBUF-1        ;ASCII码首地址减1送SI
```

```
        MOV   BX,ASCLEN        ;ASCII 码字符个数送 BX
ATOB1:  MOV   AL,[SI+BX]       ;取一个 ASCII 码(从后往前)
        AND   AX,000FH         ;转二进制数
        MUL   MULT10           ;乘当前权值
        ADD   BINVAL,AX        ;送累加和单元
        MOV   AX,MULT10
        MUL   CX               ;计算下一位的权值
        MOV   MULT10,AX        ;改变当前权值
        DEC   BX               ;计数减1
        JNZ   ATOB1            ;未处理完,转 ATOB1
        RET
ASCTOBIN ENDP
```

三、表的处理和应用

表是若干数据项组成的集合,它的应用很广泛。程序设计中常遇到对表的处理问题。一般来说,对表的处理包括查找、删除、替换、排序操作等。下面分别举例说明。

(一)查表

1. 顺序查表法

顺序查表法是指从给定的表首地址开始从前往后依次查找指定项目的方法。适合于对无序表的查找。但是,如果有序表中的数据项目比较多,则也可以采用这种查找方法进行查找。

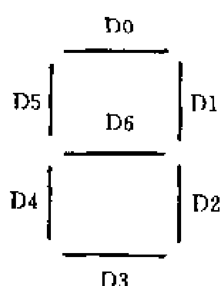


图5-18 七段显示数码管

例5.26 许多单片机系统上的显示器采用七段代码方式的数码管。如图5-18所示。现设数码管的各段由数据总线的 D0~D6位控制,若某一位为0,则相应段点亮;若某位为1,则相应段不亮。要让数码管显示数字1,应送给它代码40H;显示1,应送79H;显示2,应送24H;……显示 F,应送0EH。这些毫无规则的代码可以组成一个表,存放在内存以 SSEG 为首址的连续单元中。可用 DB 定义:

SSEG DB 40H,79H,24H,30H,…,0EH

它们对应的显示数字分别是十六进制数字0~F。如果现在任给一七段显示代码,要求利用此表求出它所对应的十六进制数字来。因为无规律可循,只好用顺序查表法。

实现它的程序段如下:

```
        MOV   AL,XX               ;设任给的代码在 XX 单元
        MOV   CX,10H              ;表长
        MOV   BX,OFFSET SSEG      ;BX 指向表首址
        MOV   AH,0                ;十六进制数字初值
LOP1:   CMP   AL,[BX]
        JE    AT
        INC   AH
        INC   BX
```

```
        LOOP   LOP1
        MOV    AL,0FFH             ;若查不到,0FFH送AL
        JMP    DONE
AT:     MOV    AL,AH               ;查到时AH中为对应的十六进制数字
DONE    HLT
```

本程序的执行结果在AL中,若表中无所给代码,则AL中为0FFH,若有,则AL中即为对应的十六进制数字,因为表中代码存放的顺序对应于0~F的顺序。

顺序查找法是所有查找方法中最简单、但也是最慢的一种方法。它的平均查找次数D = n/2。当数据个数n很大时,花费的时间就很长。

2. 对分查表法(折半法)

若表中的内容已按大小次序排列好,则可采用对分查找法。这种方法可用大大减少查找次数。对分查找的思想是:先取表中间的值 $e_{N/2}$(N/2处的值)与要查找的值X比较,看是否相等,若相等则搜索到;若不等则比较两数的大小,若 $X>e_{N/2}$,则下次取(N/2)~N中间的中间值 $e_{3N/4}$ 与X比较;若 $X<e_{N/2}$,则下次取0~N/2中间的值 $e_{N/4}$ 与X比较,这样每查找一次使区间缩小1/2,如此一直进行下去,直至或者是被搜索的字找到,或者是搜索的区间变为0,则表示搜索不到所要找的数。

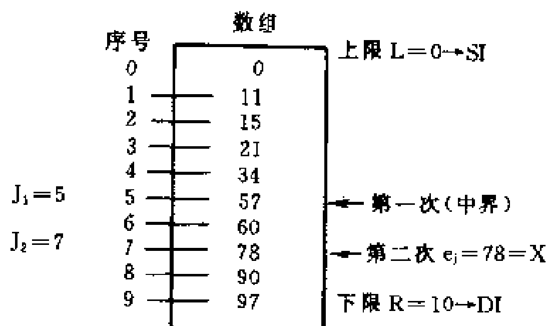


图5-19　搜索过程示意图

例5.27　有一组数00、11、15、21、34、57、60、78、90、97。数据的个数N=10,而数的排列序号为0~(N-1)=9。设要搜索的数为X=78(又称X为关键字KEY)。

取区间上限L=0,区间下限R=10,则序号 $J_1=(L+R)/2=5$ 取出的数 $e_{J1}=57$ 与X= 78相比,两者不等;而且 $X>e_{J1}$。于是下一次要到下半区间去寻找,则取 $L=J_1=5$,R=10不变,求出新的序号 $J_2=(L+R)/2=(5+10)/2=7.5$ 往下取整,取 $J_2=7$,则 $e_{J2}=78$ 与X相等,找到。上述的搜索过程可用图5-19表示。

搜索过程中,求出的序号J已与区间的上限相等时,表示搜索的区间已压至最小。若此时的 e_J 仍不等于X,则表示被搜索的数组中没有要搜索的关键字。PRINT单元及PRINT+1单元为FFH。

按上述方法进行查找的程序如下:

```
BUF_DAT SEGMENT
   BUFFER  DB 00,11,15,21,34,57,60,78,90,97
   COUNT   EQU  $-BUFFER
   PTRN    DW ?            ;存放查找次数或未找到标记
   CHAR    EQU 78          ;关键字78
BUF_DAT  ENDS
STACK   SEGMENT PARA STACK 'STACK'
```

```
  STAPN   DB 100 DUP(?)
STACK   ENDS
CODE    SEGMENT
    ASSUME  CS:CODE,DS:BUF_DAT
    ASSUME  SS:STACK,ES:BUF_DAT
  STR  PROC  FAR
    START:  PUSH  DS
            MOV   AX,0
            PUSH  AX
            MOV   AX,BUF_DAT
            MOV   DS,AX
            MOV   ES,AX
            MOV   SI,OFFSET BUFFER          ;区间上限送 SI
            MOV   CX,COUNT
            MOV   DX,1                      ;查找次数
            MOV   AX,SI
            ADD   AX,CX                     ;最后数的地址加1
            MOV   DI,AX                     ;下限送 DI
            MOV   AL,CHAR
    CONT1:  MOV   BX,SI
            ADD   BX,DI
            SHR   BX,1                      ;BX 中为 J
            CMP   AL,[BX]                   ;关键字与 ej 比较
            JZ    FOUND                     ;找到转 FOUND
            PUSHF                           ;保护标志
            CMP   BX,SI                     ;J=上限吗?
            JZ    NOFID                     ;相等未找到
            POPF
            JL    LESS                      ;关键字<ej 转
            MOV   SI,BX                     ;J 作上限
            JMP   NEXT
    LESS:   MOV   DI,BX                     ;J 作下限
    NEXT:   INC   DX                        ;查找次数加1
            JMP   CONT1
    NOFID:  MOV   DX,0FFFFH                 ;未找到标记0FFFFH
            POPF
    FOUND:  MOV   AX,DX
            MOV   PTRN,AX                   ;结果送 PTRN 单元
            RET
  STR  ENDP
```

```
CODE      ENDS
          END   START
```

当数据较多时，对分查找比顺序查找速度快得多，但它要求表的内容有序。设表长为 N=65536。顺序查找平均次数 n1 为：n1=N/2=65536/2=32768，而对分查找平均次数 n2有公式（公式证明从略）可计算：n2=LOG_2(N−1)=LOG_2(65536−1)=15。

3. 计算查表法

计算查表法不同于前述的两种查表方法，它不需要进行关键字的比较，而是通过一定的方法直接找到被查元素的存储地址，从而找到被查元素的一种方法。

例5.28 数据或程序的加密和解密

为了使数据能够保密，可以建立一个密码表，利用 XLAT 指令查表将数据加密。例如，从键盘上输入0～9间的数字，经加密后存到每次中，密码可选择为：

原数字： 0、1、2、3、4、5、6、7、8、9

密码数字：7、5、9、1、3、6、8、0、2、4该加密程序如下所示，程序接受键入的一个数字，加密后存入 MIMA 单元。

```
DATA  SEGMENT
   mitab   DB '7591368024'        ;加密密码表
   count   equ $-mitab
   jmitab  DB '7384915062'        ;解密密码表
   mina    DB ?
DATA  ENDS
CODE  SEGMENT
          ASSUME  CS:CODE,DS:DATA
   STARTPROC FAR
            PUSH  DS
            MOV   AX,0
            PUSH AX
            MOV   AX,DATA
            MOV   DS,AX
            MOV   AH,1            ;从键盘输入一个数字
            INT   21H
            AND   AL,0FH
            LEA   BX,mitab
            XLAT  mitab
            MOV   mina,AL
            MOV   DL,'-'
            MOV   AH,2
            INT   21H
            MOV   DL,mina
            MOV   AH,2            ;输出加密数字
```

```
            INT   21H
            RET
START  ENDP
CODE   ENDS
       END  START
```

可以改造上述程序为循环程序，使之能连续地接收键盘输入数字，遇到某一规定的标志字符时结束输入，并将输入的数字加密后存到内存缓冲区。

用类似的方法，可将内存中的程序代码加密，将编译或汇编后的高级语言程序或汇编语言程序加密。在数据通信中也利用用类似方法，先将要发送的代码加密以后再发送。

为将加密后的数据或程序复原，可编写解密程序。下述程序可将 MIMA 单元中的数据解密，结果送屏幕显示。

```
MOV   AL,MIMA
AND   AL,0FH
LEA   BX,JMITAB
MOV   AH,0
ADD   BX,AX
MOV   DL,[BX]
MOV   AH,6
INT   21H
HLT
```

还可以利用 XLAT 指令将键盘输入的密码数字解密。下述程序接收键盘输入一个密码数字，解密后的数字在 AL 中。

```
MOV   AH,1
INT   21H
AND   AL,0FH
LEA   BX,JMITAB
XLAT  JMITAB
HLT
```

可以用同样的方法对加密的程序或通信的数据解密。

(二)对无序表中的元素排序

例5.29　利用冒泡排序方法对内存单元 ARRAY 开始的以字为单位的数据进行排序。

冒泡排序的方法是：首先将无序表中的第一个元素和第二个元素进行比较，若第一个元素大于第二个元素，则将这两个元素进行交换，否则不交换。然后比较第二个和第三个元素，依次类推，直到第 n－1个元素和第 n 个元素进行比较、交换后为止。经过这样一趟排序，即把表中最大的元素查找出来并放于最后一个元素的位置上。然后，对前 n－1个元素进行同样的比较、交换，操作完成后则把具有次大的元素放于第 n－1个元素的位置上。重复以上过程直至没有元素交换为止。这样既得到一个数据按由小到大排序的表。下列是对有7 个元素的无序表进行冒泡排序的过程。

表的初始状态：　[49　38　65　97　76　13　27]
第一趟排序之后：[38　49　65　76　13　27] 97
第二趟排序之后：[38　49　65　13　27] 76　97
第三趟排序之后：[38　49　13　27] 65　76　97
第四趟排序之后：[38　13　27] 49　65　76　97
第五趟排序之后：[13　27] 38　49　65　76　97
第六趟排序之后：　13　27　38　49　65　76　97

下面是冒泡循环执行具体实现。在排序过程中，为了避免不必要的排序，程序设置了一个标志单元 XCHG-FLAG，其初值被置为0FFH，表示被排序的表是一个无序的表。在每一趟排序开始时，首先检测此标志，若此标志为0，则结束排序；若此复制为0FFH，则准备进行排序，并预置此标志为0，然后进行本趟的排序比较，若在本趟两两比较后，发生了交换，则重新置标志为0FFH，以便在下一趟排序时检测此标志，继续排序。若两两比较后，本趟未发生交换，则置标志为0，表示数据已排好序，从而使下一次排序不再进行。

程序清单：

```
STACK     SEGMENT PARA STACK 'STACK'
          DB 128 DUP(0)
STACK     ENDS
DATA      SEGMENT
     ARRAY        DW 49,38,65,97,76,13,27
     WOTD-COUNT DW ($-ARRAY)/2          ;数据个数(以字为单位)
     XCHG-FLAG   DB 0FFH                ;交换标志单元
DATA      ENDS
CODE      SEGMENT
          ASSUME CS:CODE,DS:DATA,ES:DATA,SS:STACK
     SORT    PROC FAR
             PUSH   DS
             XOR   AX,AX
             PUSH   AX
             MOV   AX,DATA
             MOV   DS,AX
             MOV   ES,AX
     RETRY:CMP   XCHG-FLAG,0
             JE   EXIT                  ;上次未发生交换,转 EXIT
             MOV   XCHG-FLAG,0          ;预置交换标志单元为0
             DEC   WORD-COUNT
             MOV   CX,WORD-COUNT        ;本趟比较次数→CX
             MOV   SI,OFFSET ARRAY      ;数据首地址→SI
             CLD
     MEXT1:LODSW                        ;取一个数据→AX
             CMP   AX,[SI]              ;与下一个数据进行比较
```

```
            JLE   NEXT2                  ;前一数据小,转 NEXT2
            XCHG  [SI],AX
            XCHG  [SI-2],AX              ;后一数据小,进行交换
            MOV   XCHG-FLAG,0FFH         ;置交换标志为0FFH
     NEXT2: LOOP  NEXT1                  ;循环进行两两数据的比较
            JMP   SHORT RETRY            ;转 RETRY,继续进行下一趟排序
     EXIT:  RET                          ;排序完成,返回 DOS
     SORT   ENDP
CODE  ENDS
        END  SORT
```

思考题与习题

1. 写出完成下列要求的变量定义语句:

(1)为某缓冲区 BUFF 预留240个字节的存储空间。

(2) 将字符串'BYTE'、'WORD'存放于某数据区。

2. 以图示说明下列语句实现内存分配和预置数据:

```
VAR1  DB 12,-12H,3 DUP(0,FFH)
VAR2  DB 100 DUP(0,2 DUP(1,2),0,3)
VAR3  DB 'HELLO.COM'
VAR4  DW VAR+6
VAR5  DD VAR3
```

3. 80X86汇编语言程序中段的类型有几种,各段如何定义?段定义中,定位类型、组合类型、类别各起什么作用,各有什么含义?

4. 代码段中,开始的一段程序有通用性,试将此程序定义为一条宏指令。

5. 请定义一条宏指令,它可以实现任一数据块的传送(假设无地址重叠),只要给出源和目标数据块的首地址以及数据块的长度即可。

6. 下列语句中,哪些是无效的汇编语言指令?并指出无效指令中的错误。

```
MOV   SP,AL
MOV   WORD_OP[BX+4×3][SI],SP
MOV   VAR1,VAR2
MOV   CS,AX
MOV   DS,BP
MOV   SP,SS:DATA_WORD[SI][DI]
MOV   AX,VAR1+VAR2
MOV   AX,[BX-SI]
INC   [BX]
MOV   25,[BX]
MOV   [8-BX],25
```

7. 若 x、y、z 已定义为字节变量，若 x 和 y 各存放一个32 位的无符号数(存放顺序是低位字节在先)，试写出将 x 和 y 相加后结果存入 z 的程序段。

8. 若题7中 x、y 各存放一个32位的有符号数(低字节数在前)，试编写 x－y 且结果存入 z 的程序段，同时判断运算结果是否发生溢出，若不溢出使 DL 清零，否则(溢出)以－1作为标志存入 DL 中。

9. 若数组 ARRAY 在数据段中已作以下定义：

```
ARRAY   DW 100 DUP(?)
```

试指出下列语句中各操作符的作用，指令执行后有关寄存器产生了什么变化：

```
 ⋮
MOV   BX,OFFSET ARRAY
MOV   CX,LENGTH ARRAY
MOV   SI,0
 ⋮
ADD   SI,TYPE ARRAY
```

10. 某软件共可接收10个键盘命令(分别为 A,B,C,…J)，完成这10个命令的程序分别为过程 P0,P1,…P9。编程从键盘接收命令，并转到相应的过程去执行。要求用两种方法：

(1)用比较、转移指令实现。

(2)用跳转表实现。

11. 假设已定义以下数据段：

```
DATA   SEGMENT
   BUF     DB 100 DUO(?)
   GOOD    DB ?
   PASS    DB ?
   BAD     DB ?
   AVRG    DB ?
     ⋮
DATA   ENDS
```

若已将某年级100名学生电路分析课的成绩以压缩型 BCD 码形式存入变量 BUF 中，试编写程序段统计成绩高于85分、低于60分和界于60分至85分的学生人数，仍以压缩型 BCD 码形式存入 GOOD,BAD 和 PASS 变量中(假定任一档的人数都不达到100人)，并计算其平均成绩，也以压缩型 BCD 码形式存入变量 AVRG 中(假定平均成绩低于100分，且舍去小数点后的数)。

12. 若 ARRAY 和 MAX 都定义为字变量，并在 ARRAY 数组中存放了10个16位无符号数，试编写程序段，找出数组中最大数，并存入变量 MAX 中。

13. 若题12的 ARRAY 数组存放的是有符号数，试编写相应的程序段。

14. 试编写一程序段，完成两个以压缩型 BCD 码格式表示的16位十进制数的加法运算，相加的两数 x 和 y 可定义为字节变量，并假定高位在前，和数 SUM 也同样定义为字节变量。

15. 对题15，若16位十进制数以非压缩型的 BCD 码格式存放，试编写相应的程序段。

16. 编写一个统计 AX 中1的个数的程序段，统计结果存放在 CL 中。

17. 编写一个判断 AX 中的数是正数、负数还是零的程序段。若(AX)≤-1，以-1存入 CL；(AX)＝0，以0存入 CL；否则若(AX)＞0，以1存入 CL。

18. 假定有一个由100个元素组成的字节数组(且是无符号数)，该数组已在数据段中定义为字节变量 TABLE，试编写一段程序，把出现次数最多的数存入 CH 中，其出现次数存入 CL 中。

19. 假定有一最大长度为80个字符的字符串已定义为字节变量 STRING，试编写一程序段，找出第一个空格的位置(00H 至4FH 表示)，并存入 CL 中，若该串中无空格符，则以-1存入 CL 中。

20. 对题19，若该字符串以回车符结束，试编写一段程序，统计该串的实际长度(不包括回车符)，统计结果存入 CL 中。

21. 编写统计 AX 中0、1个数的程序。0的个数存入 CH，1的个数存入 CL 中。

第六章 存 储 器

第一节 概 述

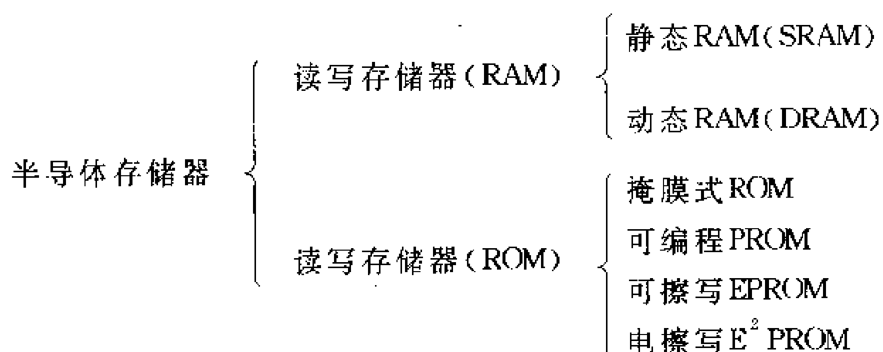
存储器把要处理问题的程序和所需的原始数据存储起来，处理时CPU自动而连续地从存储器中取出程序中的指令并执行指令规定的操作。中间数据也利用存储器保存起来。这就是说，计算机每完成一条指令，至少有一次为取指令而访问存储器的操作。上面谈到的这种存储器是内存储器，也称主存储器，或简称存储器。内存储器用来存放当前机器运行的程序和数据。它是计算机主机的一部分，一般把具有一定容量的速度较高的存储器作为内存储器，CPU可直接用指令对内存储器进行读写。在微机中，通常是用半导体存储器作为内存储器。

另一类存储器是存储容量大、速度较低、位于主机之外的存储器，称为外存储器或海量存储器。它用来存放当前暂时不用的程序和数据。CPU不能直接用指令对外存储器进行读写。要使用外存储器中的信息，必须先将它调入内存储器，在微机中常用硬磁盘、软磁盘和磁带作为外存储器。

本章仅对作为内存储器的半导体存储器的讨论。

一、半导体存储器的分类

半导体存储器按存取方式不同，分为读写存储器RAM和只读存储器ROM 。读写存储器(RAM-Random Access Memory)指可以随机地、个别地对任意一个存储单元进行读、写的存储器，可按信息存储方式分为静态RAM(SRAM-Static RAM)和动态RAM(DRAM-Dynamic RAM)。只读存储器(ROM-Read Only Memory)指在正常工作情况下只能读出信息，而不能写入信息的存储器，可分为掩膜式ROM(MROM)，一旦生产厂家做好，不能更改；可编程ROM(PROM-Programmable ROM)可以写入一次，写入后不能更改；可擦除PROM(EPROM-Erasable PROM)可多次写入，写入后可用紫外线擦除重新写入；电擦除PROM(E^2PROM-Electrically Erasable PROM)。



二、半导体存储器的主要技术指标

1、速度指标：存取时间和存储周期

存取时间是指从启动一次读出或写入操作到完成该操作所需要的时间，一般为几百纳秒。存储周期是指连续启动两次独立的操作所需的最小间隔时间。可知，存储周期略大于存取时间。

2、存储容量：存储器所能存储的二进制信息总数。有 3 种表示法：

(1)所能存储的总字数。

(2)字数×字长＝存储元数，即可存储二进制信息的总位数。

(3)能存储字节的总数。

存储器容量与地址线数的关系如表 6-1 所示。

表 6-1　存储容量与存储单元的关系

存储容量	1K	2K	4K	8K	16K	32K	64K	128K	256K	512K	1M
存储单元数	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288	1048576
2^n	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}

3、可靠性

可靠性一般是指外界电磁场干扰及温度的变化对存储器的影响。半导体存储器受温度影响较小。

4、功耗与集成度

目前半导体存储器多采用 NMOS 工艺制作，其功耗较低，典型值约为 0.1mW/ 位。若采用 CMOS 工艺制作，每位功耗可降低到 μW 数量级，集成度要高。

第二节　读写存储器(RAM)

一、静态 RAM(SRAM)

1、基本存储电路

该电路通常由如图 6-1 所示的 6 个 MOS 管组成。

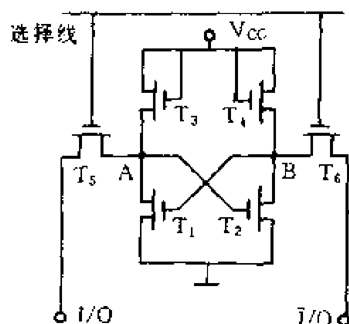


图 6-1　六管静态 RAM 存储电路

在此电路中，T_1～T_4 管组成双稳态触发器，T_1、T_2 为放大管，T_3、T_4 为负载管，若 T_1 截止，则 A 点为高电平，使 T_2 导通，于是 B 点为低电平，保证 T_1 截止。同样，T_1 导通而 T_2 截止，这是另一个稳定状态。因此可用 T_1 管的两种状态表示"1"或"0"。由此可知 SRAM 保存信息的特点是和这个双稳态触发器的稳定状态密切相关的。显然，仅仅能保持这两个状态还是不够的，还要对状态进行控制，于是加上了控制管 T_5、T_6。

当地址译码器的某一个输出线送出高电平到 T_5、T_6 控制管的栅极时，T_5、T_6 导通，于是，A 与I/O线相连，B 点与$\overline{I/O}$线相连。这时如要写"1"，则 I/O

为“1”，$\overline{I/O}$为“0”，即 A= “1”，B=“0”，使 T_1 截止，T_2 导通。而当写入信号和地址译码信号消失后，T_5、T_6 截止，该状态仍能保持。如要写“0”，$\overline{I/O}$线为“1”，I/O 线为“0”，这使 T_1 导通，T_2 截止，只要不掉电，这个状态会一直保持，除非重新写入一个新的数据。

对所存的内容读出时，仍需地址译码器的某一输出线送出高平到 T_5，T_6 管栅极，则此存储单元被选中，此时 T_5、T_6 导通，于是 T_1、T_2 管的状态被分别送至 I/O，$\overline{I/O}$线，这样就读取了所保存的信息。显然，所存储的信息被读出后，所存储的内容并不改变，除非重写一个数据。

由于 SRAM 存储电路中，MOS 管数目多，故集成度较低，而 T_1、T_2 管组成的双稳态触发器必有一个是导通的，功耗也比 DRAM 大，这是 SRAM 的两大缺点。其优点是不需要刷新电路，从而简化了外围电路。

2、芯片结构

静态 RAM 内部是由很多如图 6-1 所示的基本存储电路组成的。容量为单元数与数据线位数之乘积。为了选中某一个单元，往往利用矩阵式排列的地址译码电路。例如 1K 单元的内存，需 10 根地址线，其中 5 根用于行译码，另 5 根用于列译码，译码后在芯片内部排列成 32 条行选择线和 32 条列选择线，这样可选中 1024 个单元中的任何一个。而每一个单元的基本存储电路个数与数据线位数相同。

常用的典型 SRAM 芯片 Intel 6116 的引脚及功能框图如图 6-2 所示。

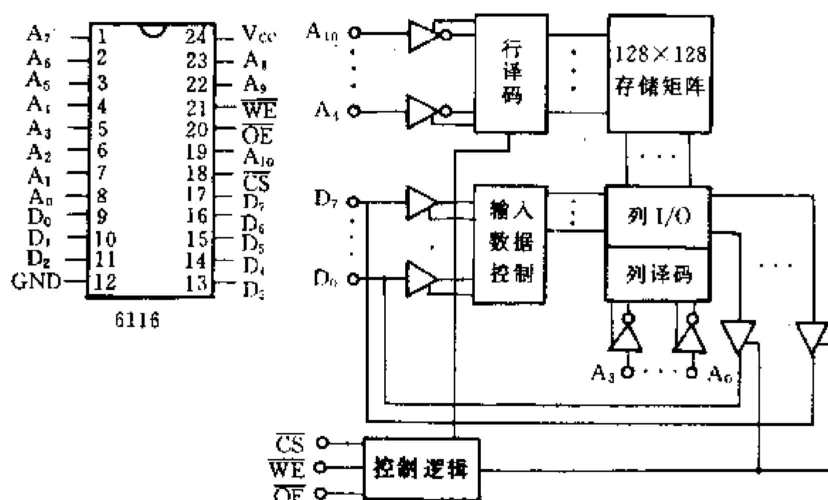


图 6-2　6116 引脚和功能框图

6116 芯片的容量为 2K×8 位，有 2048 个存储单元，需 11 根地址线，7 根用于行地址译码输入，4 根用于列地址译码输入，每条列线控制 8 位，从而形成了 128×128 个存储阵列，即存储体中有 16384 个存储元。6116 的控制线有三条：片选$\overline{CS}$、输出允许$\overline{OE}$和读写控制$\overline{WE}$。

Intel 6116 存储器芯片的工作过程如下：

读出时，地址输入线 A_{10}～A_0 送来的地址信号经译码器送到行、列地址译码器，经译码后选中一个存储单元（其中有 8 个存储位），由$\overline{CS}$，$\overline{OE}$，$\overline{WE}$构成读出逻辑（$\overline{CS}$=0，$\overline{OE}$=0，$\overline{WE}$=1），打开右面的 8 个三态门，被选中单元的 8 位数据经 I/O 电路和三态门送到 D_7～D_0 输出。

写入时，地址选中某一存储单元的方法和读出时相同，不过这时$\overline{CS}$=0，$\overline{OE}$=1，$\overline{WE}$=0。打开左边的三态门，从 D_7～D_0 端输入的数据经三态门的输入控制电路送到 I/O 电路，从而写到存储单元的八个存储位中。

当没有读写操作时，$\overline{CS}=1$，即片选处于无效状态，输入输出三态门呈高阻状态，从而使存储器芯片与系统总线“脱离”。

二、动态 RAM(DRAM)

在动态 RAM 中，存储信息的基本电路可以采用四管电路、三管电路和单管电路。由于基本电路使用的元件数目减少，因而集成度可进一步提高。目前多利用单管电路来作为存储器基本电路。

1、单管动态基本存储电路

单管动态基本存储电路，如图 6-3 所示，数据信息存储在电容 C_1 上，该 C_1 是 MOS 管栅极与衬底之间的分布电容。若 C_1 上存有电荷，表示信息为“1”，否则为“0”。而由三管或四管组成的一个基本存储电路，也是靠 MOS 管栅极与衬底之间分布电容来记忆信息的。虽然 MOS 管是高阻器件，漏电流小，但漏电流总还是存在的，因此 C_1 上的电荷经一段时间就会泄放掉(一般约为几毫秒)，故不能长期保留信息，为了维持动态存储单元所存储的信息，必须进行刷新，使信息再生。

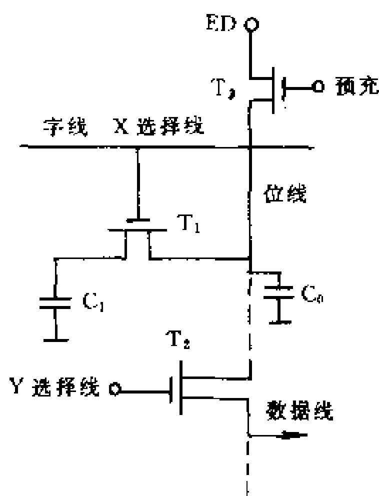


图 6-3 单管动态基本存储电路

MOS 管 T_1 作为一个开关，T_2 管为同一列基本存储电路所共用的，C_0 为位线上对地的寄生电容。写数时 X 选择线、Y 选择线置 1。此时 T_1、T_2 导通，外部信息通过数据线加至位线，然后再通过 T_1 加至 C_1 上。若数据线上为高电平，则 C_1 充电至高电平；若数据线上为低电平，C_1 通过 T_1、T_2 放电至低电平。

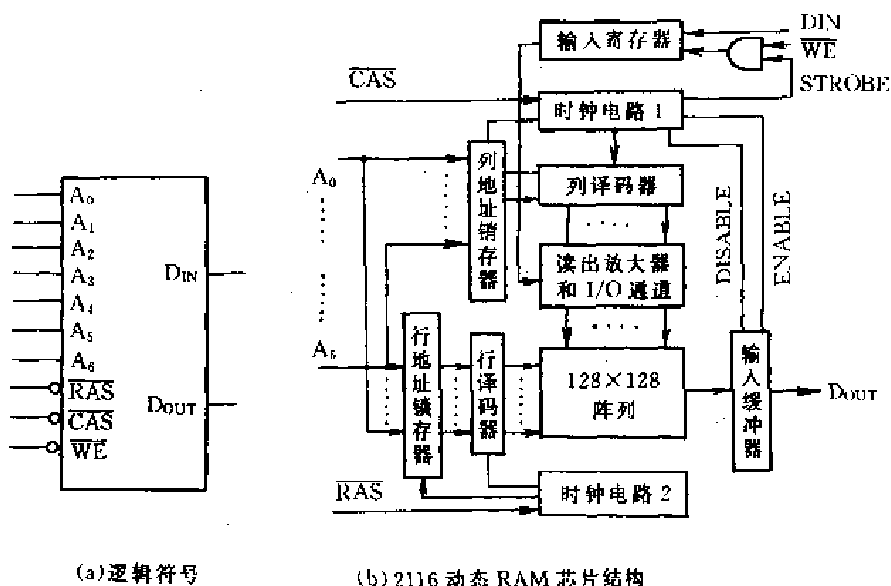
读数时，先要进行预充电，T_3 管栅极加一正的预充电脉冲，使 C_0 上电压为 1，而后 X、Y 选择线置 1，此时 T_1 导通，C_1 上的电压传至位线。由于电容 C_1 很小，约为 0.1～0.2pF，所以读出的 0 与 1 信号的差很小，需要进行放大。另外在每次读出后，由于 C_1 上的电荷发生了变化，原先的存储内容受到破坏，因而还必须把原来信号重新写入。刷新过程就是读出信息(不送到数据线上)经放大后再传送给位线进行写入的过程。

2、芯片结构

常用的典型的 DRAM 芯片 Intel 2116 的逻辑符号和芯片结构如图 6-4 所示。Intel 2116 芯片容量为 16K 位，采用位结构方式组成 163384 位的形式，有 A_0～A_6 7 条地址输入端，一条 D_{in} 数据输入端，一条数据输出端 D_{out}，$\overline{RAS}$行地址选通端，$\overline{CAS}$列地址选通端，写允许输入端$\overline{WE}$。

为了访问 16K 存储空间，需要 14 根地址线($2^{14}=16384$)。但 2116 芯片封装在 16 脚管壳内。其引脚数较少，实际使用时将地址线分成两部分：7 位行地址和 7 位列地址。7 位行地址选择 128 行，7 位列地址选择 128 列。但行地址和列地址之间又如何区别呢？由图 6-4(a) 逻辑符号可知，在 2116 芯片控制端有两个地址选通信号$\overline{RAS}$和$\overline{CAS}$，$\overline{RAS}$作为行地址通选信号，当该信号为低电平时，加在地址端的 7 位地址作为行地址送至芯片内部行地址锁存器保存；$\overline{CAS}$作为列地址选通信号，当该信号为低电平时，加在地址端的 7 位地址作为列地址送至列地址锁存器保存。由于动态存储器的刷新是一行一行进行的，因而 7 位刷新地址，由$\overline{RAS}$信号选通。

3、动态 RAM 的刷新



(a)逻辑符号　　(b)2116 动态 RAM 芯片结构

图 6-4　2116 DRAM 芯片的逻辑符号和结构框图

当动态 RAM 与 CPU 连接时，为了访问某一存储单元，CPU 将该存储单元的 14 位地址由地址寄存器加到地址总线。在刷新过程中还需接入刷新地址，为了分别选通行地址，列地址和刷新地址，需要外加多路转换器，其具体连接如图 6-5 所示。

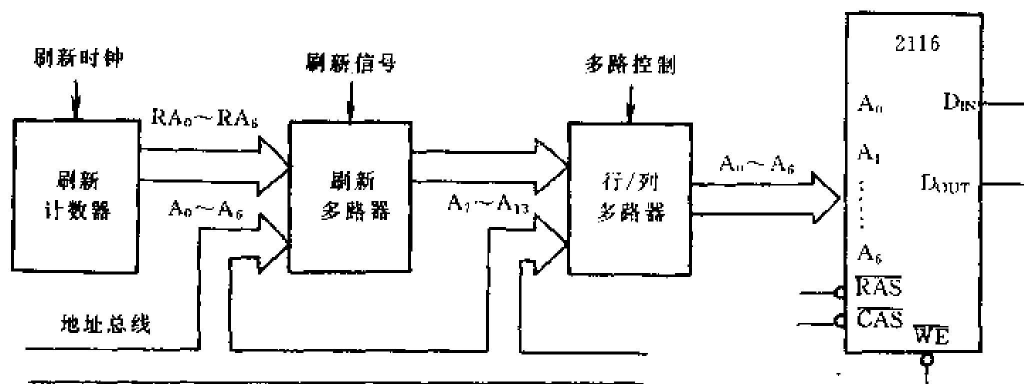


图 6-5　动态 RAM 与存储器、控制器连接框图

由图 6-5 可知，地址总线上的 $A_0 \sim A_6$ 作为行地址，7 位行地址和刷新计数器的输出 $RA_0 \sim RA_6$，均加到刷新多路器的输入端，平时 7 位行地址通过刷新多路器输出，只有在刷新时 $RA_0 \sim RA_6$ 才能作为刷新地址输出。刷新多路器的输出加在行/列多路器的输入端，地址总线上 $A_7 \sim A_{13}$ 作为 7 位列地址也加在行/列多路器的输入端。在工作过程中 7 位行地址先通过行/列多路器，加到 2116 芯片的地址输入端，由行选通信号 $\overline{RAS}$ 将 7 位行地址送行地址锁存器保存，随后 7 位列地址通过行/列多路器再由列选通信号 $\overline{CAS}$ 将 7 位列地址送到列地址锁存器，待行地址和列地址信号稳定后，即可选中某一存储单元进行读出和写入操作。

在刷新过程中选通信号 $\overline{RAS}$ 为低电平，而 $\overline{CAS}$ 为高电平，此时刷新地址作为行地址送入动态 RAM。每一个刷新地址使存储矩阵行中所有存储元(在本列中有 128 个基本元)在一个周期内同时刷新。

动态 RAM 除了进行读写操作外，还要定时进行刷新操作以保证存储器正常工作。刷新方式有以下 3 种：

(1)在几毫秒时间内每隔一段时间刷新一次。以 2116 为例，在 2ms 时间内要刷新 128 行，若每隔 15μs 刷新一行，则在 1.92ms 时间内可将 128 行轮流刷新一遍。

(2)在 2ms 时间内集中一段时间进行刷新操作，在这段时间内存储器不能进行读写操作，将这段时间称为死时间。

(3)在每一个指令周期中利用 CPU 不进行访内操作的时间进行刷新。

第三节 只读存储器(ROM)

只读存储器 ROM 的信息在使用时是不能被改变的，即只能读出，不能写入，故一般只能存放固定程序和常量，如监控程序，IBM PC 中的 BIOS 程序等。ROM 的特点是非易失性的，即掉电后再上电时存储信息不会改变。ROM 芯片种类很多，下面介绍其中的几种：

一、掩膜式 ROM(MROM)

掩膜式 ROM 制成后，用户不能修改，图 6-6 为一个简单的 4×4 位 MOS 管 ROM，采用单译码结构，两位地址线 A_1～A_0 译码后可译出四种状态，输出 4 条选择线，可分别选中 4 个单元，每个单元有 4 位输出。

图中所示的矩阵中，在行和列的交点，有的连有管子，有的没有，这是工厂根据用户提供的程序对芯片图形(掩膜)进行二次光刻所决定的，所以称为掩膜式 ROM。

若地址线 A_1A_0＝00，则选中 0 号单位，即字线 0 为高电平，若有管子与其相连(如位线 2 和 0)，其相应的 MOS 管导通，位线输出为 0，而位线 1 和 3 没有管子与其相连，则输出为 1。故存储器的内容取决于制造工艺，图 6-6 存储矩阵的内容如表 6-2 所示。

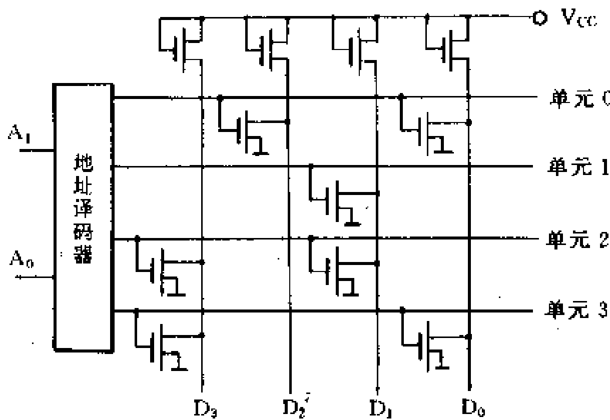


图 6-6 掩膜式 ROM 示意图

表 6-2 掩膜式 ROM 的内容

单元 \ 位	D3	D2	D1	D0
0	1	0	1	0
1	1	1	0	1
2	0	1	0	1
3	0	0	1	0

二、可编程只读存储器(PROM)

为了便于用户根据自己的需要来确定 ROM 中的内容，出现了可编程的只读存储器，简称为 PROM。它可以由用户自己编程。

图 6-7 是一种 32×8 的熔丝式 PROM 结构图，每一个字为 8 位，共 32 个字。每一个字的 8 位，实际上是一个多发射极(8 个)管，每一个发射极通过一个熔丝与位线相连。管子工作在射

极输出器状态，当它被选中时，基极为高电位，故熔丝连着的位经过读写控制电路反相输出为"0"。若熔丝烧断，则位线就不与管子的射极相连，经读写控制电路反相输出为"1"。出厂时所有管子的熔丝都是连着的，可由用户根据需要把某些熔丝烧断，相当于存入"1"信息；未烧断的则相当于存入"0"信息。

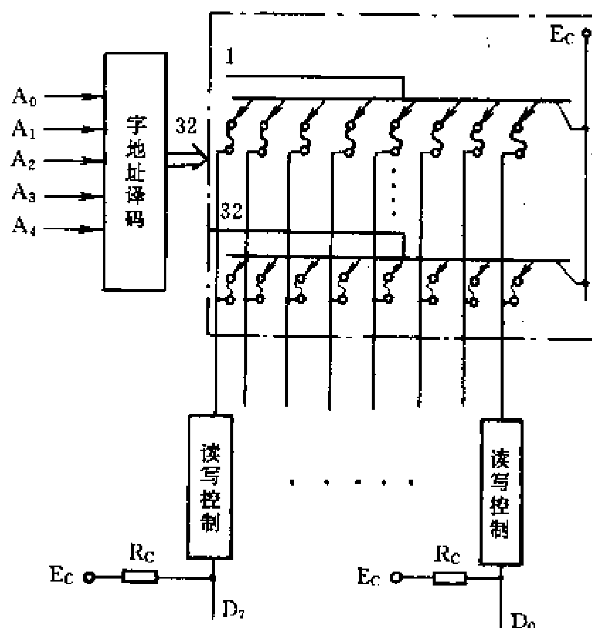


图 6-7 一种 32×8 熔丝式 PROM

三、可擦写只读存储器(EPROM)

在某些应用中，程序需要经常修改，因此能够重复擦写的 EPROM 被广泛应用。这种存储器利用编程器写入后，信息可长久保持，因此可作为只读存储器。当其内容需要变更时，可利用擦抹器(由紫外线灯照射)将其擦除，各基本元内容复原(为 FFH)，再根据需要利用 EPROM 编程器编程，因此这种芯片可反复擦写。

1、基本存储电路

通常 EPROM 存储电路是利用浮栅 MOS 管构成的，又称 FSMOS 管(Floating gate Avalanche injection Metal-Oxide-Semiconductor，即浮栅雪崩注入 MOS 管)，其构造如图 6-8 所示。

该电路和普通 P 沟道增强型 MOS 管相似，只是栅极没有引出端，而被 SiO_2 绝缘层所包围，称为"浮栅"。在原始状态，栅极上没有电荷，该管没有导通沟道，D 和 S 是不导通的。如果将源极和衬底接地，在衬底和漏极形成的 PN 结上加一个约 24V 的反向电压，可导致雪崩击穿，产生许多高能量的电子，这样的电子比较容易越过绝缘薄层进入浮栅。注入浮栅的电子数量由所加电压脉冲的幅度和宽度来控制。如果注入的电子足够多，这些负电子在硅表面上感应出一个连接源、漏极的反型层，使源漏极呈低阻态。当外加电压取消后，积累在浮栅上的电子没有放电回路，因而在室温和无光照的条件下可长期的保存在浮栅中。将一个浮栅管和 MOS 管串起来组成如图 6-8(b)所示的基本存储电路。于是浮栅中注入了电子的 MOS 管源、漏极导通，当行选线选中该存储元时，相应的位线为低电平，即读取值为"0"，而未注入电子的浮栅管的源、漏极是不导通的，故读取值为"1"。在原始状态，即厂家出厂时，没有经过编程，浮栅中没有

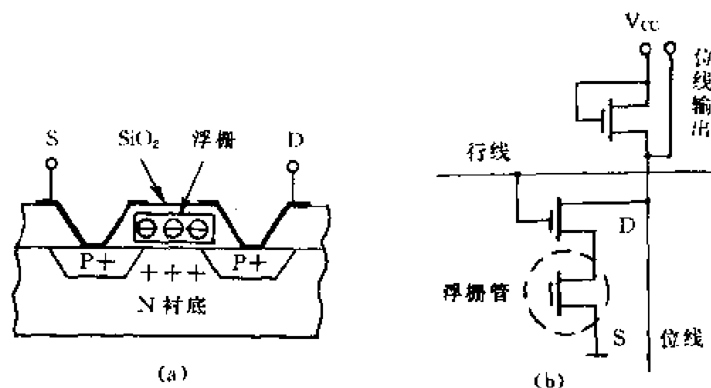


图 6-8 浮栅 MOS EPROM 存储电路

注入电子，位线上总是“1”。

消除浮栅电荷的办法是利用紫外线光照射，由于紫外线光子的能量较高，从而可使浮栅中的电子获得能量，形成光电流从浮栅流入基片，使浮栅恢复初态。EPROM 芯片上方有一个石英玻璃窗口，只要将此芯片放入一个靠近紫外线灯管的小盒中，一般照射 20 分钟，读出各基本存储电路的内容均为 FFH，则说明该 EPROM 已擦除。

2、EPROM 芯片介绍

EPROM 芯片有多种型号，如 2716(2K×8)、2732(4K×8)、2764(8K×8)、27128(16K×8)、27256(32K×8)等。下面以 2764A 为例，对 EPROM 的性能和工作方式作一介绍。

Intel 2764A 有 13 条地址线，8 条数据线，2 个电压输入端 Vcc 和 Vpp，一个片选端 $\overline{CE}$(功能同 $\overline{CS}$)，此外还有输出允许 $\overline{OE}$ 和编程控制端 $\overline{PGM}$，其功能框图见图 6-9。

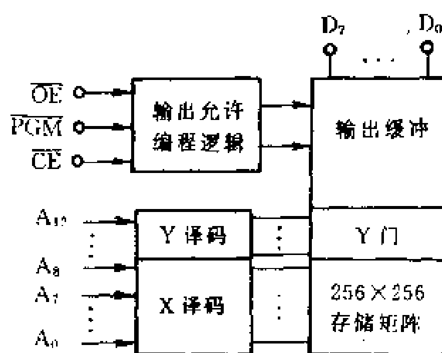


图 6-9 2764A 功能框图

Intel 2764A 有七种工作方式，如表 6-3 所示。

表 6-3 2764A 工作方式选择表

方式 \ 引脚	$\overline{CE}$	$\overline{OE}$	$\overline{PGM}$	A_9	A_0	V_{PP}	V_{CC}	数据端功能
读	低	低	高	×	×	Vcc	5V	数据输出
输出禁止	低	高	高	×	×	Vcc	5V	高阻
备用	高	×	×	×	×	Vcc	5V	高阻
编程	低	高	低	×	×	12.5V	Vcc	数据输入
校验	低	低	高	×	×	12.5V	Vcc	数据输出
编程禁止	高	×	×	×	×	12.5V	Vcc	高阻
标识符	低	低	高	高	低	Vcc	5V	制造商编码
					高	Vcc	5V	器件编码

(1)读方式

这是 2764A 通常使用的方式，此时两个电源引脚 Vcc，Vpp 都接至 5V，$\overline{PGM}$ 接至高电平，

当从 2764A 的某个单元读取数据时，先通过地址引脚接收来自 CPU 的地址信号，然后使控制信号$\overline{CE}$和$\overline{OE}$都有效，于是经过一个时间间隔，指定单元的内容即可读到数据总线上。

(2)备用方式

只要$\overline{CE}$为高电平，2764A 就工作在备用方式，输出端为高阻状态，这时芯片功耗将下降，从电源所取电流由 100mA 下降到 40mA。

(3)编程方式

这时，Vpp 接 12.5V，Vcc 仍接 5V，从数据线输入这个单元要存储的数据，$\overline{CE}$端保持低电平，输出允许信号$\overline{OE}$为高，每写一个地址单元，都必须在$\overline{PGM}$引脚端给一个低电平有效，宽度为 45ms 的脉冲，如图 6-10 所示。

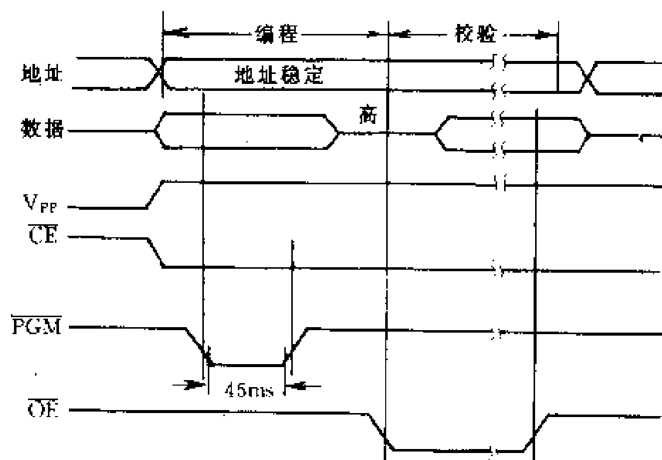


图 6-10　2764A 编程时的波形

(4)编程禁止

在编程过程中，只要使该片$\overline{CE}$为高电平，编程就立即被禁止。

(5)编程校验

在编程过程中，为了检查编程时写入的数据是否正确，通常在编程过程中包含校验操作。在一个字节编程完成后，电源的接法不变，但$\overline{PGM}$为高电平，$\overline{CE}$、$\overline{OE}$均为低电平，则同一单元的数据就在数据线上输出，这样就可与输入数据相比较，校验编程的结果是否正确。

(6)Intel 标识符模式

当两个电源端 Vcc 和 Vpp 都接至 5V，$\overline{CE}=\overline{OE}=0$ 时，$\overline{PGM}$为高电平，这时与读方式相同，但把 A_9 引脚接至 11.5～12.5V 的高电平，则 2764A 处于读 Intel 标识符模式。

因为标识符是两个字节，要读出 2764A 的编码必须顺序读出两个字节，先让 A_1～A_8 全为低电平，而使 A_0 从低变高，分两次读取 2764A 的内容，当 $A_0=0$ 时读出的内容为制造商编码(陶瓷封装为 89H，塑封为 88H)，当 $A_0=1$ 时，则可读出器件的编码(2764A 为 08H，27C64 为 07H)。

可利用编程器对 EPROM 编程，编程器中有一个卡插在 I/O 扩展槽上，外部接有 EPROM 插座，所提供的编程软件可自动提供编程电压 Vpp，按菜单提示，可读、可编程、可校验、也可读出器件的编码，操作很方便。

3、高集成度 EPROM

除了常使用的 EPROM2764 外，还常使用 27128、27256、27512 等。由于工业控制计算机的发展，迫切需用电子盘取代硬盘，常把用户程序、操作系统固化在电子盘(ROMDISK)上，这时

要用 27C010(128K×8)、27C020(256K×8)、27C040(512K×8)等大容量芯片。关于这几种芯片的使用请参阅有关手册。

四、电擦写可编程只读存储器(E^2PROM)

E^2PROM 是近年来开始被推广应用的只读存储器，其主要特点是能在应用系统中进行在线读写，并在断电情况下保存的数据信息不会丢失。因此 E^2PROM 存储器具有广泛的应用前景。

1、E^2PROM 的应用特性

(1)对硬件电路没有特殊要求，操作十分简单。早期产品如 2816、2817 是依靠片外高压电源(约 20V)进行擦除的。后来把高压电源集成在片内，构成了新型 E^2PROM 芯片。如 2816A、2817A、2864A 等。给用户带来了极大方便，省去了电路中高压电源。

(2)采用+5V 电擦写 E^2PROM，是在写入过程中自动进行擦写的。但目前擦写时间较长，约需 10ms 左右，需要保证有足够的写入时间。有的 E^2PROM 芯片设有写入结束标志，可供查询或中断使用。

(3)E^2PROM 存储器除了有并行传送数据芯片外，还有串行传送数据芯片，串行 E^2PROM 具有体积小、成本低、电路连接简单，占用系统地址线和数据线少的优点，但数据传送速度低。

2、E^2PROM 芯片介绍

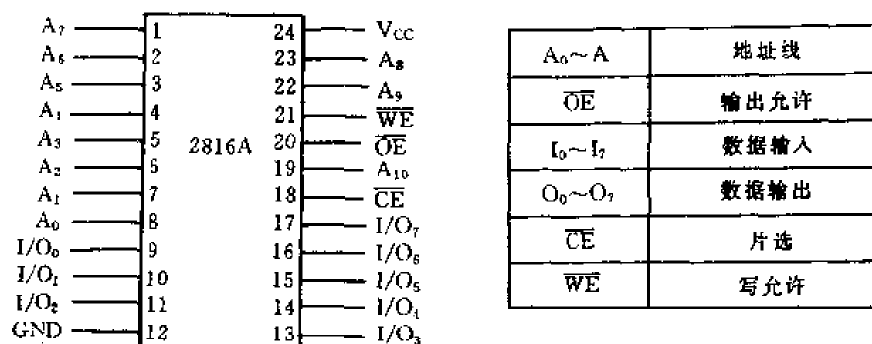
(1)Intel 公司 E^2PROM 典型产品主要性能如表 6-4 所示，表中列出了 2816、2817、2816A、2817A 及 2864A 的主要性能。

表 6-4　几种 E^2PROM 产品的主要性能

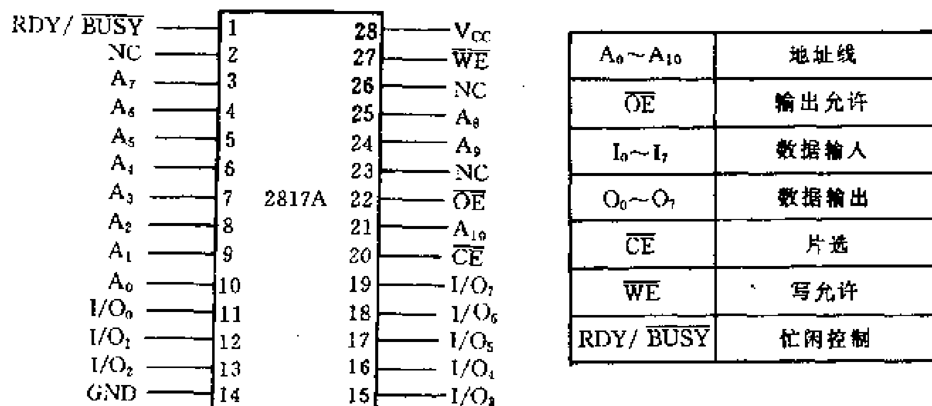
器件型号	单位	2816	2816A	2817	2817A	2864A
取数时间	ns	250	200/250	250	200/250	250
读操作电压 V_{PP}	V	5	5	5	5	5
写/擦操作电压 V_{PP}	V	21	5	21	5	5
字节擦写时间	ms	10	9-15	10	10	10
写入时间	ms	10	9-15	10	10	10
封装		DIP24	DIP24	DIP28	DIP28	DIP28

(2)E^2PROM2816A、2817A 引脚与工作方式

2816A 与 2817A 均为 2K×8 字节的 E^2PROM，不需要外加高压电源，连线简单，使用方便。但 2817A 有擦写完毕信号端 $RDY/\overline{BUSY}$ 在擦写操作期间，$RDY/\overline{BUSY}$ 为低电平，当字节擦写完毕时，$RDY/\overline{BUSY}$ 引脚为高电平。E^2PROM2816A 与 2817A 如图 6-11 所示。而 2816A 工作方式如表 6-5 所示，2817A 工作方式如表 6-6 所示。



(a)E²PROM2816A



(b)E²PROM2817A

图 6-11　E²PROM 2816A，2817A 引线图

表 6-5　2816A 工作方式

工作方式＼引脚	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	输入/输出
读	0	0	1	D_{OUT}
维持	1	任意	任意	高阻
字节擦除	0	1	0	$D_{IN}=1$
字节写入	0	1	0	D_{IN}
全片擦除	0	−10V～+15V	0	$D_{IN}=V_{IN}$
不操作	0	1	1	高阻
E/W 禁止	1	1	0	高阻

表 6-6　2817A 工作方式

工作方式＼引脚	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	RDY/$\overline{BUSY}$	输入/输出
读	0	0	1	高阻	D_{OUT}
维持	1	任意	任意	高阻	高阻
字节写入	0	1	0	D_{OL}	D_{IN}
全片擦除	字节写入前自动擦除				

第四节 存储器的组织

一、存储器的结构

1、存储体

存储器是由大量的基本存储电路组成。这些存储电路有规则地组合起来就成为存储体。在较大容量的存储器中，往往把各个字的同一位组织在一个片中，这样的存储芯片称为多字一位片，如 256K×1 位，512K×1 位等；也有把各个字的几位组织在一个片中，称多字多位片，如 256K×4 位、1K×4 位等。

图 6-12 是一个典型的 RAM 芯片结构示意图，它的存储体是 1024×1，即 1024 个字的同一位。不同字的同一位通常排成矩阵的形式，如 32×32＝1024，这是为了便于译码寻址。

2、外围电路

如图 6-12 所示，一个存储器芯片除了存储体外，还有外围电路，通常有：

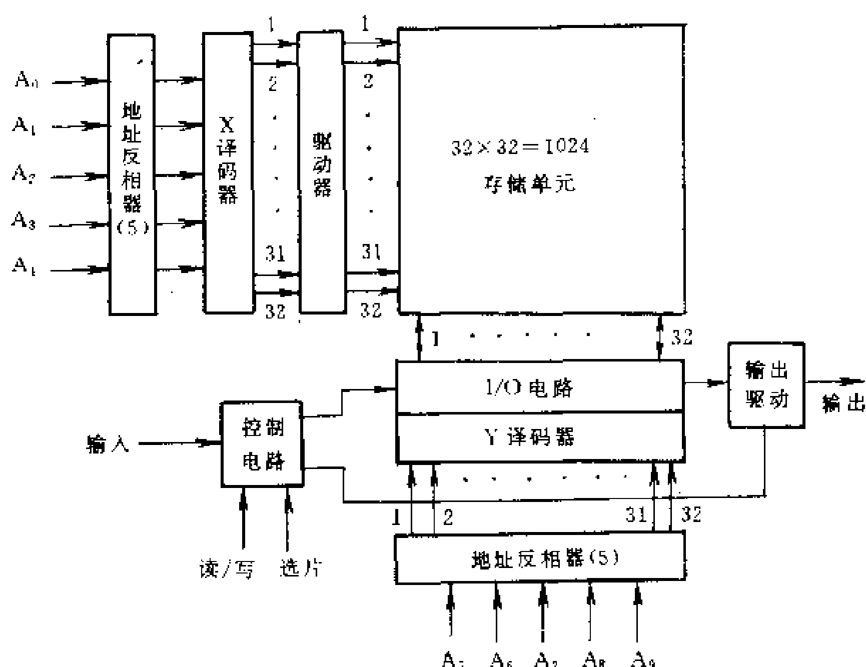


图 6-12 典型的 RAM 示意图

(1)地址译码器。用以对 n 条地址线译码，以选择 2^n 个存储单元中的一个。

(2)I/O 电路。处于数据总线和被选用的单元之间，用以控制被选中的单元读出或写入，并具有驱动作用。

(3)片选控制端 $\overline{CS}$(Chip Select)。由于每一片芯片的存储容量总是有限的，所以，一个存储器往往由一定数量的片子组成。在地址选择时，首先要选片，用地址译码器输出和一些控制信号(如 8086 的 M/$\overline{IO}$)形成选片信号，只有当某一片的 $\overline{CS}$ 输入信号有效，该片所连的地址线才有效，才能对这一片上的存储单元进行读或写的操作。

(4)集电极开路或三态输出缓冲器。为扩展存储器的字数，常需将几片 RAM 的数据线并联使用，或与双向的数据总线相接，因而需要用到集电极开路或三态输出缓冲器。

另外，在动态 MOS 型 RAM 中，还有预充、刷新等方面的控制电路。

3、地址译码方式

存储器芯片的地址译码有两种方式：一种是单译码方式，又称字结构，适用于小容量的存储器芯片；另一种是双译码，或称重合译码结构。

(1)单译码结构。图 6-13 是一种单译码结构的存储器芯片示意图。为了说明问题，我们假设它只是一个 16 字 4 位的存储器，并且把它排成 16 行×4 列，则每一行对应一个字，每一列对应其中的一位。每一行选择线和每一列的数据线是公共的。在这种结构中，n 根地址输入经全译码有 2^n 个输出，用以选择 2^n 个字，例如 16 个字对应 $A_0 \sim A_3$ 共 4 根地址线，经译码获得 24=16 根选择线。显然，随着存储字数的增加，译码的输出线数及相应的驱动电路会急剧增加，存储器成本也将迅速增加。

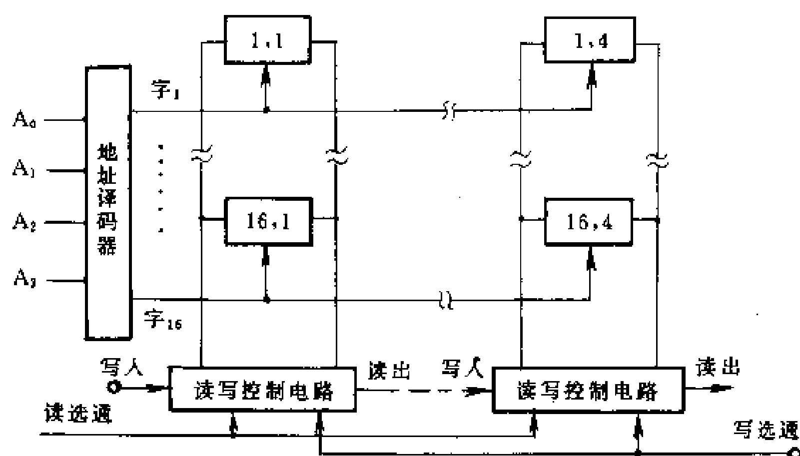


图 6-13　单译码结构存储器

(2)双译码结构。双译码结构往往用于地址位数 n 很大时，这时把 n 位地址线分成接近相等的两段，分别译码，产生一组 X 地址线和一组 Y 地址线，然后让 X 地址线和 Y 地址线在存储元排成矩阵形式的存储体中一一相"与"，选择相应的存储元。

图 6-14 给出了一个有 1K(1024)个字的存储器的双译码电路。1024 个字排成 32×32 的矩阵，10 根地址线分成 $A_0 \sim A_4$ 和 $A_5 \sim A_9$ 两组。前组经 X 译码器输出 32 条行选择线，后组经 Y 译码器输出 32 条字选择线。行选择线和字选择线的组合可以方便地找到 1024 个中的任何一个，而译码器输出的总线数仅为 $2^5+2^5=64$ 根，而不是采用单译码时的 $2^{10}=1024$ 根。

图 6-15 给出了一个 1K×4 位的 SRAM Intel 2114 的结构方框图。它的 10 根地址线中的 $A_3 \sim A_8$ 用于行译码，A_0、A_1、A_2 和 A_9 用于列向的字选择。存储器的内部数据通过 I/O 电路以及输入和输出的三态门与数据总线相连。三态门受控于片选信号 $\overline{CS}$ 与写允许信号 $\overline{WE}$ 的组合。当 $\overline{CS}$ 有效时(为低电平)，$\overline{WE}$ 为低电平，输入三态门导通，信号由数据线写入存储器；$\overline{WE}$ 为高电平时(相当于读控制)，输出三态门打开，从存储器读出的信号送至数据线。

4、存储器的连接

对存储器进行读写操作，首先由地址总线给出地址信号，然后发出进行读或写操作的控制信号，最后在数据线上进行信息交换。因此，存储器的连接，要完成(1)地址线的连接；(2)数据线的连接；(3)控制线的连接。目前生产的存储器芯片的容量是有限的，它在字数和字长方面与实际存储器的要求都有很大差距，所以需要在字向和位向两方面进行扩充才能满足实际存储器的容量要求。在此，讨论存储器连接时地址线和数据线的连接问题。

(1)位扩展法

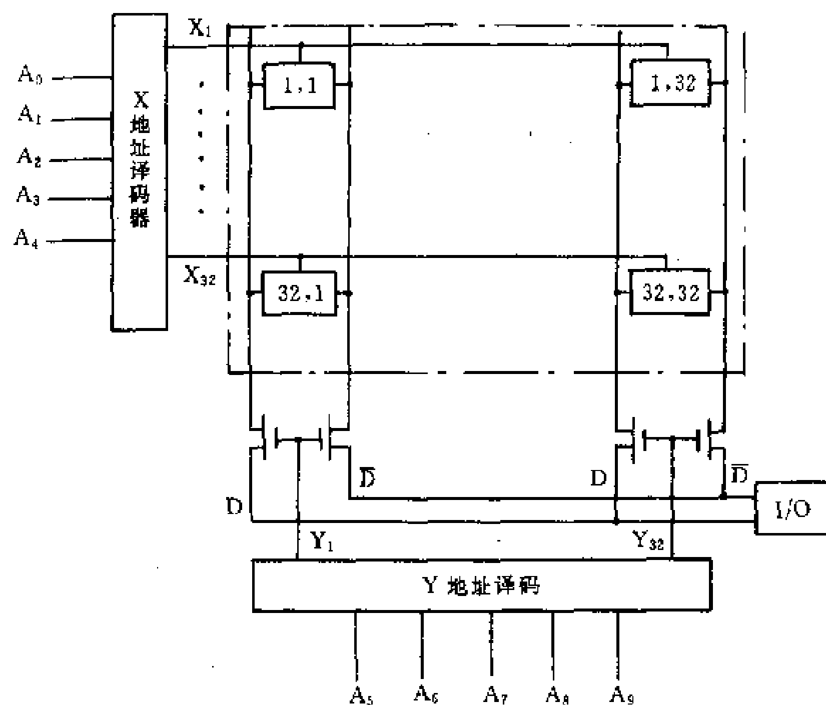


图 6-14　双译码存储器电路

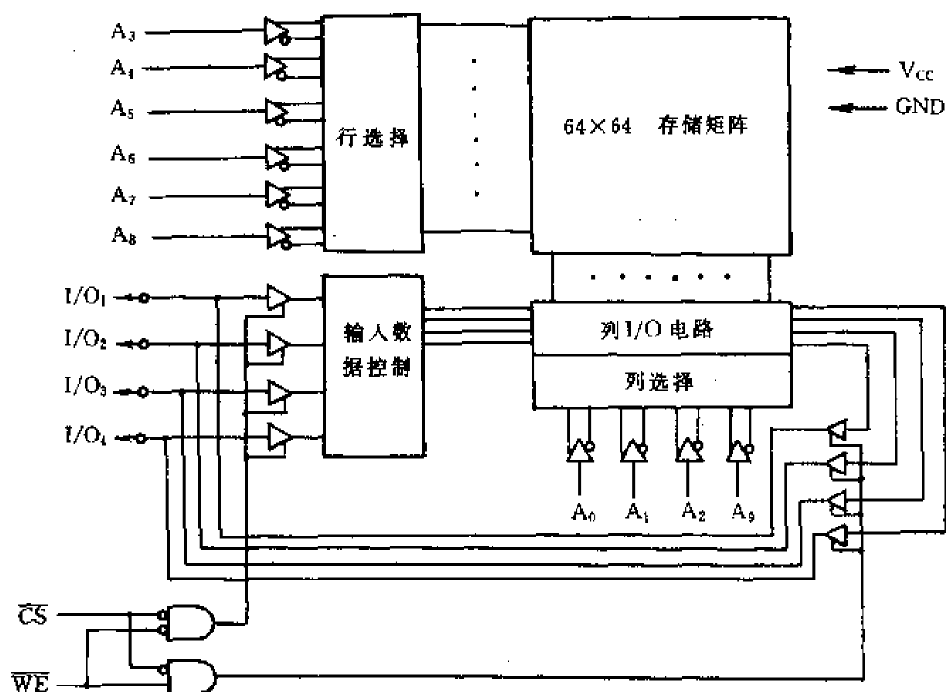


图 6-15　2114 的结构方框图

假定使用 8K×1 的 RAM 存储器芯片，那么组成 8K×8 位的存储器可采用图 6-16 所示的位扩展法。此时只加大字长，而存储器的字数与存储器芯片字数一致。图中，每一片 RAM 是 8192×1，故其地址为 13 条（A_0～A_{12}），可满足整个存储容量的要求。每一片对应于数据的 1 位（只有 1 条数据线），故只需将它们分别接到数据总线上的相应位即可。在这种方式中，对片选信号均按已被选中来考虑。有选片输入端（$\overline{CS}$），可将它们直接接地。在这种连接时，每一条

地址总线接有 8 个负载，每一条数据线接有一个负载。

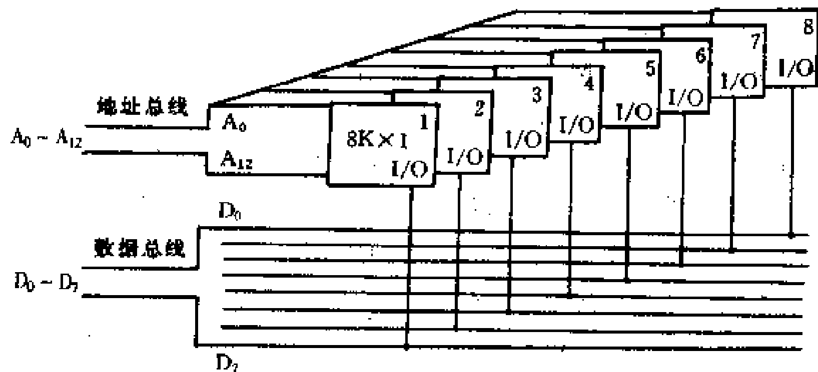


图 6-16 位扩展法组成 8K RAM

(2) 字扩展法

字扩展是只在字向扩充，而位数不变，因此将芯片的地址线、数据线、读/写控制线并联，而由片选信号来区分各片地址，故片选信号端连接到选片译码器的输出端。图 6-17 表示出用 16K×8 位的芯片采用字扩展法组成 64K×8 位的存储器连接图。图中 4 个芯片的数据端与数据总线 $D_0 \sim D_7$ 相连，地址总线低位地址 $A_0 \sim A_{13}$ 与各芯片的 14 位地址端相连，而两位高位地址 A_{14}、A_{15} 经 2-4 译码器分别与 4 个片选端相连。这四片的地址空间分配见表 6-7。

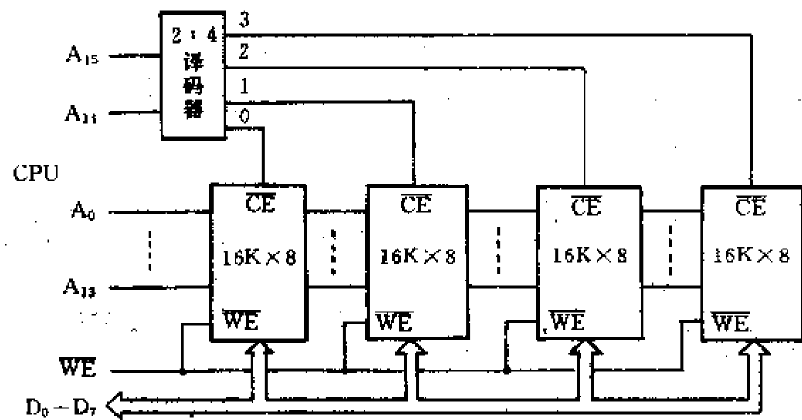


图 6-17 字扩展法组成 64K RAM

表 6-7 地址空间分配表

片号 \ 地址	$A_{15}A_{14}$	$A_{13}A_{12}A_{11}\ldots A_1A_0$	说明
1	0 0	0 0 0 ... 0 0	最低地址
	0 0	1 1 1 ... 1 1	最高地址
2	0 1	0 0 0 ... 0 0	最低地址
	0 1	1 1 1 ... 1 1	最高地址
3	1 0	0 0 0 ... 0 0	最低地址
	1 0	1 1 1 ... 1 1	最高地址
4	1 1	0 0 0 ... 0 0	最低地址
	1 1	1 1 1 ... 1 1	最高地址

(3) 字位同时扩展法

一个存储器的容量假定为 M×N 位，若使用 l×k 位的芯片（l<M，k<N），需要在字向和位向同时进行扩展。此时共需要(M/l)×(N/k)个存储器芯片。

以 2114SRAM 构成 4K×8 存储器模块为例，如图 6-18 所示。若其中某一芯片$\overline{CS}$有效，则由写允许信号$\overline{WE}$规定该片执行读操作还是写操作。若$\overline{CS}$无效，则$\overline{WE}$信号对该片不起作用，其数据输入/输出端呈高阻状态。这样就可以把同一行的 4 个 2114 芯片的相应数据输入/输出端直接连接在一起提供数据字节的 4 位。每一行构成 4K×4 RAM，两行构成 4K×8 存储器模块。采用这种办法时，每一行中哪一个芯片被选中，取决于哪个芯片的$\overline{CS}$信号有效，芯片中哪个存储单元被选中，则取决于 $A_0 \sim A_9$ 提供的地址码。阵列中的同一列芯片的$\overline{CS}$端都接到同一个列选通线上，该线由高位地址（本例中是 A_{11} 和 A_{10}）控制，如果选中某一列，该列上的两片 2114 中对应于 $A_0 \sim A_9$ 地址码的存储单元都被选中，根据$\overline{WE}$状态决定进行写操作还是读操作。例如，如果地址有 16 位，则 $A_{15} \sim A_{12}$ 用来选择存储器模块，A_{11} 和 A_{10} 来选择列，$A_9 \sim A_0$ 用来选择该列的芯片中对应存储单元。若该存储器模块占用的存储器地址为 4000H～4FFFH，则地址译码电路如图 6-19 所示。

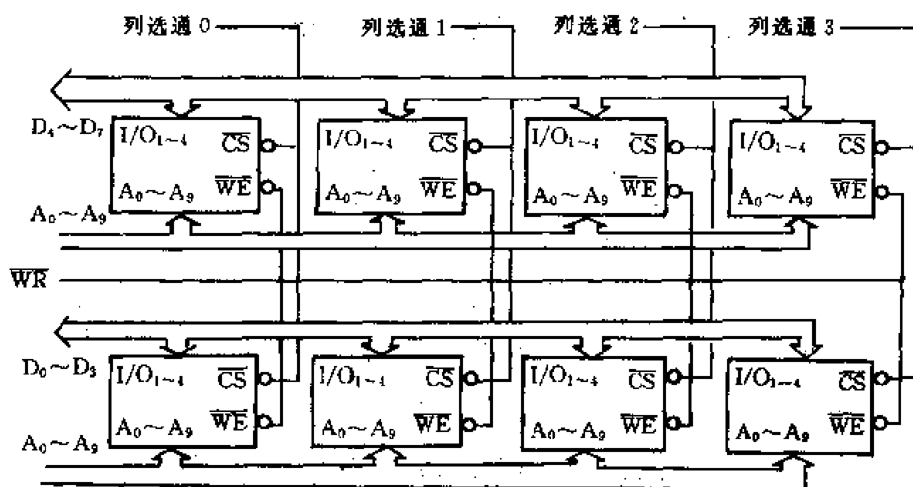


图 6-18 由 1K×4 SRAM 构成的 4K×8 存储器模块

二、8086 系统的存储器组织

1、8086CPU 的存储器接口

在 8086 最小模式系统和最大模式系统中，8086CPU 可寻址的最大存储空间为 1 兆字节。但是，8086 最小模式系统和最大模式系统的配置是不一样的。8086 最大模式系统中增设了一个总线控制器 8288 和一个总线仲裁器 8289，因而 8086CPU 和存储器系统的接口在这两种模式中是不同的。

图 6-20 是 8086 最小模式系统的存储器接口框图。寻址存储单元的信号由多路复用的地址/数据总线 $AD_{15} \sim AD_0$、地址线 $A_{19} \sim A_{16}$ 和总线高位有效信号$\overline{BHE}$提供。存储器的控制信号 ALE、$\overline{RD}$、$\overline{WR}$、M/$\overline{IO}$、DT/$\overline{R}$和$\overline{DEN}$直接由 8086CPU 产生。

图 6-21 是 8086 最大模式的存储器接口框图，该电路包括了一片 8288 总线控制器芯片。

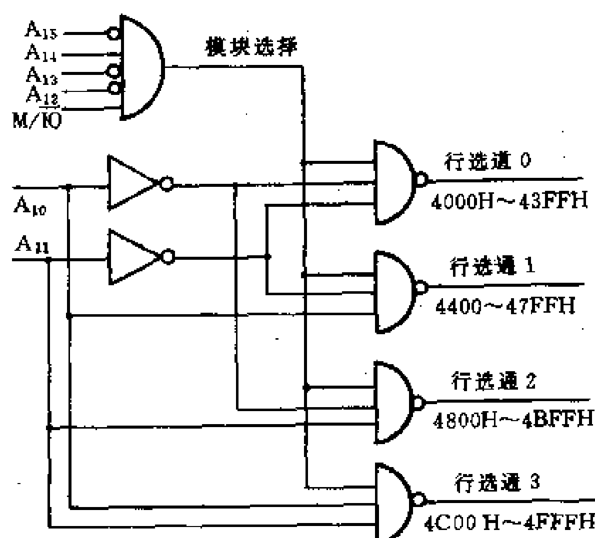


图 6-19　地址译码电路

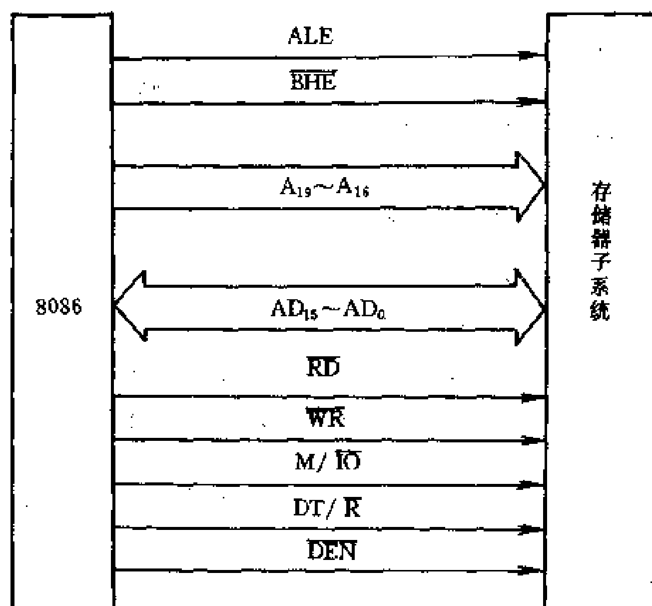


图 6-20　8086 最小模式系统存储器接口

8086 给 8288 发送总线状态信息，8288 将这三位标识总线周期类型的状态信号译码，产生了读/写信号 $\overline{MRDC}$、$\overline{MWTC}$和$\overline{AMWC}$以及控制信号 ALE、DT/$\overline{R}$和$\overline{DEN}$。由此可见，在最大模式系统中，8288 代替了 8086 产生和存储器接口的总线命令和总线控制信号，仅$\overline{BHE}$和$\overline{RD}$信号仍然由 8086CPU 提供。

在 8086 存储器系统中，20 位地址总线(A_{19}~A_0)的最大寻址存储空间是 2^{20}=1048576(1M)字节。其地址范围为 00000~FFFFFH。显然，在 8086 微型计算机系统中，存储器系统实际上是以字节为单位组成的一维线性空间。

我们在介绍 8086 存储器的组成时指出，8086 寻址的 1M 存储空间可以分成两个 512K 字节的存储体，一个存储体包含偶数地址，另一个存储体包含奇数地址。任何两个连续的字节可以作为一个字来访问，显然其中一个字节必定来自偶地址存储体，另一个必定来自奇地址存储体。地址值低的字节是低位字节，地址值高的字节是高位字节。

为了有效地使用存储空间，一个字可以存储在以偶地址或奇地址开始的连续两个字节单元中。地址的最低有效位 A_0 决定了字的边界。如果 A_0 是 0，则字存放在偶地址边界上，其低 8 位字节存储于偶地址单元中，高 8 位字节存储于相邻的奇地址单元中。同理，如果 A_0 是 1，则字是存放在奇地址边界上。

对所有位于偶地址边界上的字的访问，8086 只需一个总线周期就能完成；而对于在奇地

址边界上的字的访问，8086需要花两个总线周期才能实现。

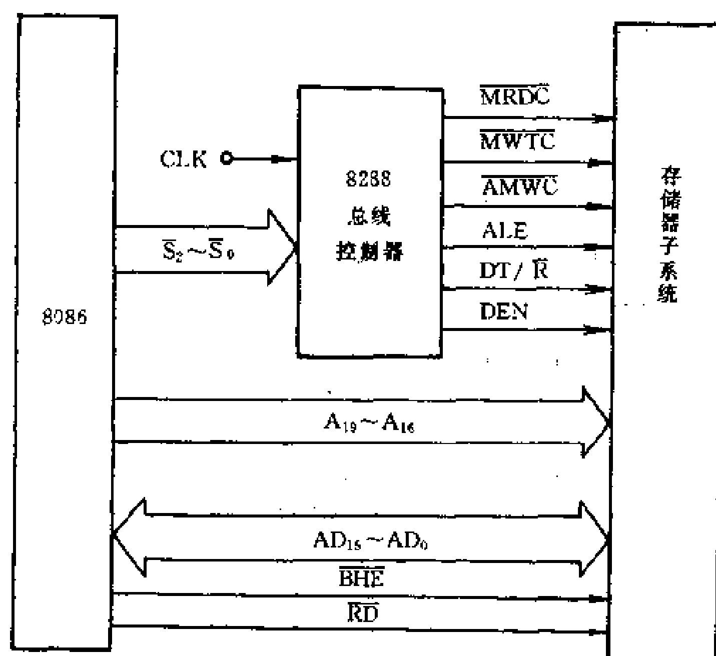


图 6-21 8086最大模式系统存储器接口

8086的1M存储空间安排如图6-22所示。从图中可知，1M存储空间的最高和最低区域是留给某些特殊的处理功能使用的。如存储单元00000～0007FH共128个字节用于存放Intel保留的32种中断矢量；FFFF0～FFFFFH共16个字节用于存放启动程序。8086应用系统不能把这些区域改作其它用途，否则会使系统与未来的Intel的产品不兼容。除此以外，ROM和RAM可位于1M存储空间的任何位置。

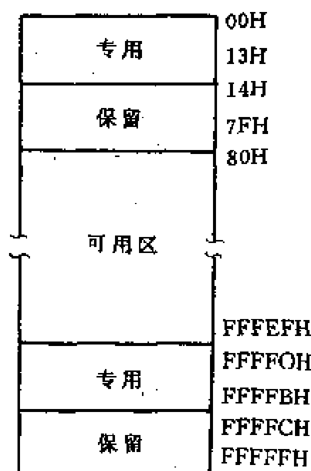


图 6-22 8086存储空间

8086在硬件结构上是如何保证自由地对奇偶两个存储体进行操作的呢？图6-23为8086存储器系统的硬件组织框图。地址A_{19}～A_1是体内地址，它们并行地连接到两个存储体上；A_0和$\overline{BHE}$用来作为存储体的选择信号，它们的组合可以保证8086自由地对两个存储体进行操作。A_0的低电平信号表示寻址数据的偶地址字节单元，允许低位存储体和低8位数据总线交换信息；$\overline{BHE}$有效（低电平）允许高位存储体和高8位数据总线交换信息。当$\overline{BHE}$有效和A_0为0时，8086同时访问两个存储体，读/写一个字的信息；当$\overline{BHE}$有效和A_0为1时，8086只访问奇地址存储体，读/写高8位字节的信息；当$\overline{BHE}$无效和A_0为0时，8086只访问偶地址存储体，读/写低8位字节的信息。

2、8086 CPU与存储器系统的连接

我们介绍了8086 CPU的存储器接口，当8086 CPU与存储器系统实际连接时，还要考虑许多问题。例如：

CPU的负载能力：CPU总线在设计时负载能力都有一定限制，一般可驱动一个TTL门。在小型系统中，CPU可以直接与存储器相连，而在较大的系统中，必须增加缓冲器、驱动器等。

CPU的时序和存储器的存取速度之间的配合问题：系统中，CPU的读写时序是固定的，这时就要考虑对存储器存取速度的要求；若存储器已经确定，则需要考虑是否要插入等待周期T_w。比如8086的主频采用5MHz，则1个时钟周期为200ns。将每个时钟周期称为1个T状态。CPU和存储器交换数据，或者从存储器取出指令，必须执行1个总线周期，而最小总线周期由4个T状态组成。如果存储器速度比较慢，CPU就会根据存储器送来的“未准备好”信号（READY信号无效），在T_3状态后插入等待状态T_W，从而延长了总线周期。

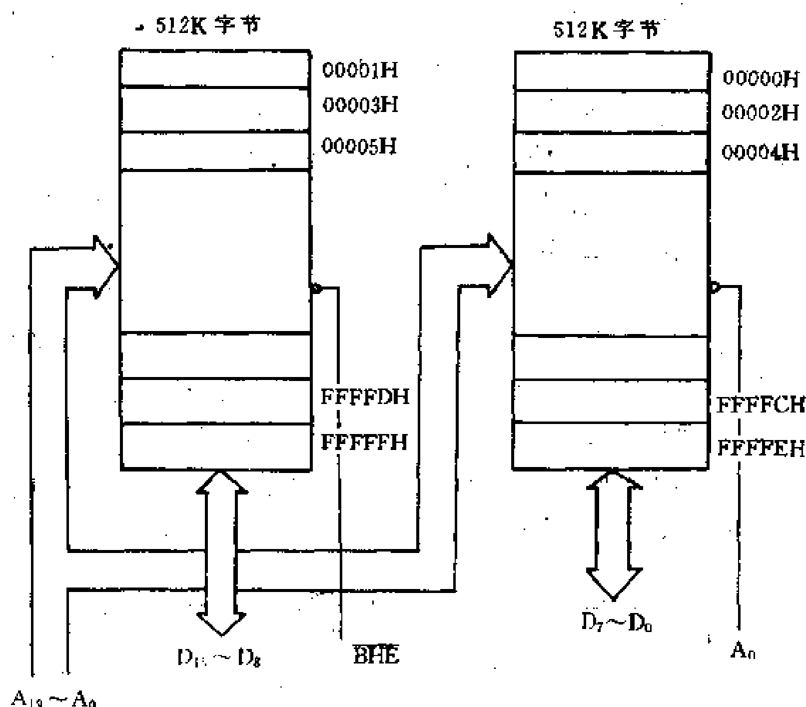


图 6-23　8086 存储器硬件组织框图

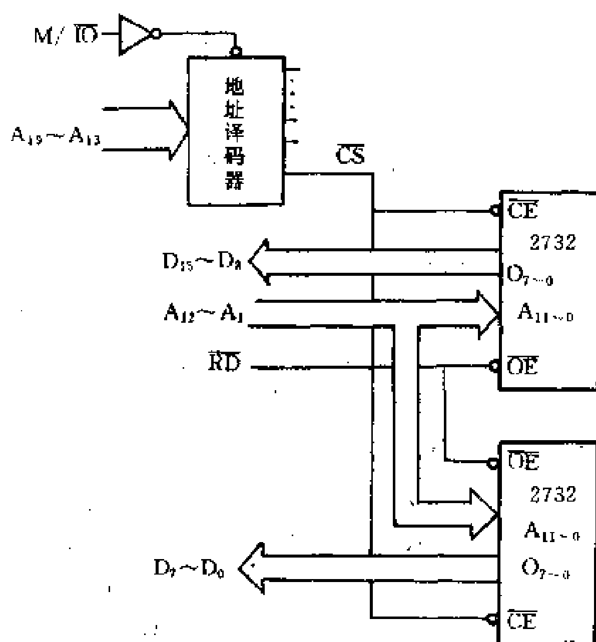


图 6-24　4K 字程序存储器

存储器的地址分配和选片问题：内存的扩展和因不同用途的分区都涉及到存储器的地址分配和选择。另外，当多片存储器芯片组成存储器时，还有一个选片信号的问题。

(1)8086 CPU 与只读存储器的连接

ROM、PROM 或 EPROM 芯片可以和 8086 系统总线连接，但是要注意像 2716、2732、2764 一类的 EPROM 芯片，它们是以字节宽度输出的，因此要用两片这样的存储芯片为单位，才能存储 8086 的 16 位指令字。

图 6-24 是两片 2732EPROM 和 8086 系统总线的连接图。该存储器子系统提供了 4K 字(8K 字节)的程序存储器(即存放指令代码的只读存储器)。图中，上面一片 2732 代表了高 8 位存储体，下面一片 2732 则代表了低 8 位存储体。为了寻址 4K 字的存储单元共需 12 条地址线(A_{12}~A_1)。两片 2732EPROM 在总线上是并行寻址的。其余的 8086 高位地址线(A_{19}~A_{13}) 用来译码产生芯片选择信号$\overline{CS}$(Chip Select)。两片 2732 的$\overline{CE}$端连接到同一个芯片选择信号。

地址译码在 CPU 与存储器的连接中是必须认真加以考虑的一个问题，特别是当实际使

用的存储空间远小于 8086 系统所能提供的寻址空间的时候,例如在图 6-24 中,地址线 A_{12}～A_1,已作为 8K 字节 ROM 的片内寻址,其余的 7 根地址线(A_{19}～A_{13})经译码器可输出 128 个片选信号线。若这 128 个片选信号线全部用上,则可寻址 128×8K=1M 字节存储器,这种地址译码称全译码方式。全译码方式显然要求用译码器,不浪费存储器地址空间,各芯片之间的地址可以连续且不致因考虑不周而使地址重复。当译码地址未用满时,扩展系统也方便。另外注意 M/$\overline{IO}$信号线在这里的作用,它可以确保只有当 CPU 要求与存储器交换数据时才会选中存储器系统。当 8086 的存储器系统较小时,可以采用地址的局部译码方式。局部译码方式只使用有限的高位地址线进行译码。例如在图 6-25(a)中,仅选 A_{15}～A_{13}作为 3:8 译码器的输入信号,译码器的 8 个片选信号共可寻址 8×8K=64K 字节存储器。显然,局部译码选择方式可以节省译码器,但比全译码选择方式的地址空间要小。特别值得注意的是,由于地址线 A_{19}～A_{16}是悬空的,因而会出现地址重叠区(地址的多重映射),这是应该避免的。图 6-25(b) 是解决地址重叠的一种方法,其原则是所有悬空的地址线都必须直接或通过附加的组合逻辑电路接入译码器,使每个存储单元仅对应一个确定的地址(实际为全译码方式).

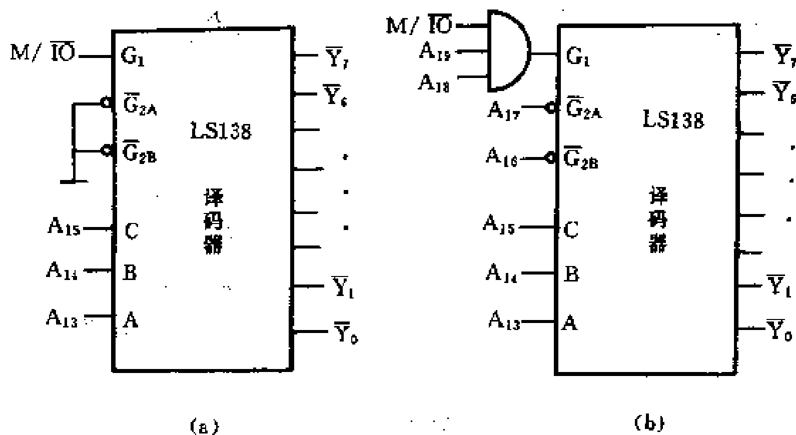


图 6-25 地址的局部译码

(2)8086 CPU 与 SRAM 连接

当微型计算机系统的存储器容量较小时(例如,少于 16K 字),宜采用 SRAM 芯片而不宜采用 DRAM 芯片。因为大多数 DRAM 芯片是位片式,如 16K×1 位或 64K×1 位。DRAM 芯片要求动态刷新支持电路,这种附加的支持电路反而增加了存储器的成本。

作为例子,图 6-26 给出了一个 1K 字的读写存储器系统。存储器芯片选用 2142 SRAM,存储器系统工作在 8086 最小模式系统中。

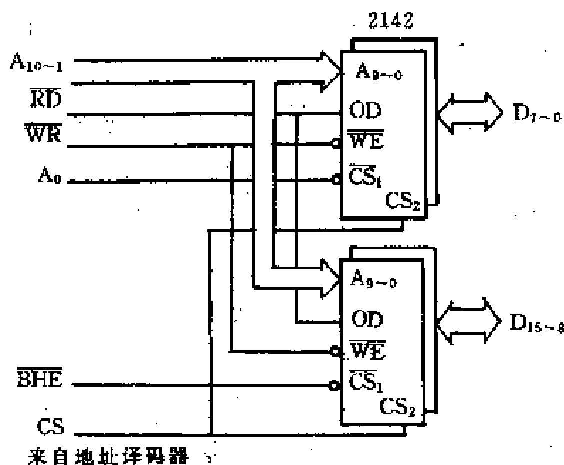


图 6-26 最小模式静态 RAM 存储器

由于 2142 静态 RAM 是以 1K×4 位组织的,所以必须用四片 2142 连接成 1K 字的数据存储器。图 6-26 实际上给出了存储器位扩展的基本方法。如用 1K×4 的芯片进行位扩展时,存储器的数不因为芯片数目增多而增加。

位扩展的连接方式是将多片存储芯片的地址、片选、读/写端相应并联，数据端单独引出。

如果要进行字向扩展，则应该把存储芯片的地址线、数据线、读/写控制线并联，而由片选信号区分各片地址，故片选端单独引出。例如要在图 6-26 的基础上进行字扩展，则新加入的奇、偶存储体的地址线、数据线和读/写控制线应该同系统中已经存在的其它存储体的相应线并联，但片选端应该连接到地址译码器的其它输出端上。

通常，存储器容量的扩展都把位向扩展和字向扩展的方法结合起来。

在图 6-26 经过位扩展的存储器系统中，每两片 2142 组成一个 1K×8 位的存储体。上面两片用作低 8 位 RAM 存储体，它们的 I/O 引线和数据总线 $D_7 \sim D_0$ 相连，代表了偶地址字节数据；下面两片 2142 用作高 8 位存储体，其 I/O 引线和数据总线 $D_{15} \sim D_8$ 相连，代表了奇地址字节数据。

地址总线 $A_{10} \sim A_1$ 并行地连接到四片 2142 的地址输入引线，用来实现片内寻址。高位地址 $A_{19} \sim A_{11}$ 和 $M/\overline{IO}$（高电平）经译码器译码产生 CS_2 的片选信号。要实现对这 1K 字存储单元的访问。高位地址部分必须引起 CS_2 片选信号为逻辑高电平，即 CS_2 必须为高电平。

$\overline{WR}$信号（低电平）用来通知存储芯片在写总线周期时数据总线上的数据已经有效。$\overline{WR}$并行地加到四片 2142 的$\overline{WE}$输入引线上，使 2142 从总线上取数据并写入所选中的存储单元。

8086 输出的$\overline{RD}$信号送到 2142 的 OD 输入线上。OD 是禁止输出的缩写，高电平 OD 禁止 2142 输出数据，低电平 OD 则允许 2142 输出数据到总线上。$\overline{RD}$信号为低电平时，允许 2142 在读总线周期时向数据总线输出数据。

A_0 和$\overline{BHE}$信号的不同组合能确保同时选中奇、偶地址体或其中之一，从而实现在数据总线传送一个字或一个字节的目的。

8086 最大模式系统中的存储器系统类似于上面描述的最小模式系统。关键的差别是在最大模式系统中，用 8288 总线控制器代替 8086 产生存储器读、写等控制信号（$\overline{MRDC}$，$\overline{MWTC}$等）。此外，系统中用 8286 总线收发器来缓冲数据总线。8288 的 $DT/\overline{R}$和DEN输出信号分别用作 8286 的 T 和$\overline{OE}$信号，用来在读/写总线周期时选择收发器的传送方向和允许数据传送。

(3)8086 CPU 与动态 RAM 连接

如果要求 8086 微型计算机系统的存储容量大于 16K 字时，存储器系统通常使用 DRAM 芯片，8086 CPU 与 DRAM 的连接比较复杂。作为一个实例，图 6-27 给出了在 8086 最大模式系统中，8086 CPU 与 128K DRAM 的连接图。

该存储器系统使用了 64 片 16K×1 位的 2118 DRAM 芯片，用来组成 128K 字节存储容量的动态存储器。8086 的地址/状态、地址/数据线经锁存器（8283）和缓冲器（8287）送至系统总线。存储器系统也使用了相应的锁存器和缓冲器来锁存、缓冲系统总线的地址和数据信息。

为了对动态 RAM 定时进行刷新，该存储器系统使用了 8202 刷新控制电路。16 位地址 $A_{16} \sim A_1$ 用作 8202 刷新控制器的地址输入（ADDR IN），地址 $A_{19} \sim A_{17}$经 8205 3-8 译码器产生刷新控制器 8202 的片选信号$\overline{PCS}$（SELECT 为低电平）。刷新控制器把输入的 16 位地址转换为 8 位行地址和 8 位列地址，由地址输出线（$\overline{ADDROUT}$）送出。8202 还产生和地址字节同步的$\overline{CAS}$和$\overline{RAS}$信号。存储体选择信号 A_0 和$\overline{BHE}$经 8202 输出的$\overline{WE}$信号选通后产生高位存储体和低位存储体写命令，以控制一个字或一个字节数据的传送。

8288 总线控制器向 8202 的$\overline{RD}$和$\overline{WR}$输入线分别提供存储器读命令$\overline{MRDC}$和存储器写命令$\overline{MWTC}$，以指示 8202 现行总线周期是读还是写总线周期，8202 则相应地向 128K 字节存储单元阵列提供$\overline{WE}$，$\overline{RAS}$，$\overline{CAS}$和$\overline{ADDROUT}$的定时信号。

图 6-27 8086 最大模式 DRAM 子系统

存储器系统的数据经过两片 8283 锁存器送到系统数据总线上，在存储器读周期，$\overline{\text{SELECT}}$和$\overline{\text{MRDC}}$信号是低电平，这两个信号经过一个与门产生 8283 锁存器的片选信号$\overline{\text{CS}}$。在$\overline{\text{CAS}}$信号作用下，DRAM在数据输出引线上送出被读单元的数据。$\overline{\text{CAS}}$返回到高电平时，产生了一个选通脉冲的下降沿，用来把存储器读出的数据装入锁存器，再送到系统数据总线上。

在存储器写周期，系统总线上的数据经 8287 总线收发器输入给存储器系统。这两片 8287 已由硬连线置成发送数据。$\overline{\text{SELECT}}$有效时，允许 8287 进行输出传送，这样，来自系统数据总线上的数据被传送到 2118 RAM 的数据输入线上。

如果 8202 控制器忙于存储器的刷新操作，则由$\overline{\text{XACK}}$信号指示，$\overline{\text{XACK}}$经缓冲后送到 8284 时钟产生器 RDY 输入端，以便 8086 在总线周期中插入等待状态。

三、80X86 存储系统简介

1. 80286 存储系统

(1) 基本存储器结构

如图 6-28 所示，80286 的总线信号与存储器的连接方法和 8086 时的情况一样，即把存储器分成偶地址的存储体和奇地址的存储体，偶地址存储体的数据线与数据总线 $D_0 \sim D_7$ 连接，而奇存储体的数据线与数据总线 $D_8 \sim D_{15}$连接。地址总线使用 $A_1 \sim A_{23}$的信号。A_0 作为偶存储体的允许信号使用。用这种连接方法，当 $A_0=0$、$\overline{\text{BHE}}=1$ 时，只是偶数地址的存储体工作。而当 $A_0=1$、$\overline{\text{BHE}}=0$ 时，只是奇数地址的存储体工作。但是，当 $A_0=0$、$\overline{\text{BHE}}=0$ 时，偶数地址的

存储体和奇数地址的存储体都同时进行工作。

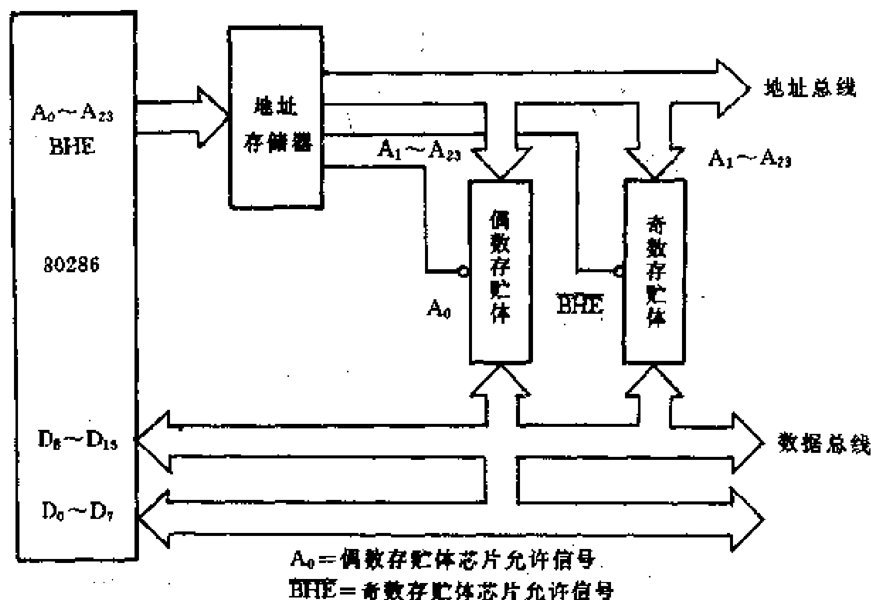


图 6-28 存储器地址总线数据总线的连接

(2) 存储器选通信号

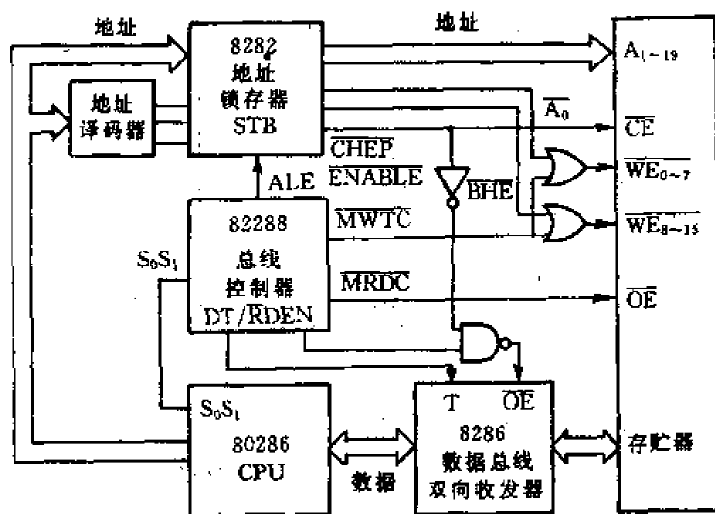


图 6-29 使用一般的选通信号的存储器

如果对图 6-28 增加命令控制信号，则成为图 6-29 的形式。图中把 82288 总线控制器输出的 ALE 信号用作地址锁存信号 STB。如图 6-30 所示，因为 ALE 信号是在 TS 状态的 Φ2 进行输出，所以，供给地址总线的地址信号，是从 TS 的 Φ2 开始，到下一个总线周期 TS 的 Φ1 结束。如果考虑到从存储器读出数据的情况，在 TC 周期结束前，能把数据输出到数据总线就可以了。因此，图中所示的 tad 是存储器响应的最大富余时间，如果使用比 tad 具有更快响应速度的存储器，对数据进行读出是不会有问题的。

因为 80286 本身是在 TS 状态以前即输出地址信号，所以在图 6-29 所示的电路中，在 TS 的 Φ2 以前输出的地址信号是无用的。如果定时电路能够提前一些供给地址锁存信号 STB，则可以使用比图 6-30 所示的 tad 响应速度更慢的存储器。在图 6-31 所示的存储器结构中，不使用 82288 输出的 ALE 信号，而把$\overline{MRDC} \cdot \overline{MWTC}=0$ 的条件用作地址锁存选通信号。这样，地址信号、$\overline{WEL}\,\overline{WEH}$信号是并行给出，而图 6-29 为串行给出。

(3) 只读存储器(ROM)

一般情况下，80286 机的系统板上的存储器为 64KB，由几片 EPROM 组成，每片 EPROM 的容量为 16KB(16K×8 位)。将 4 片 EPROM 分成两组，即形成了两个 32K × 8 位的存储体。

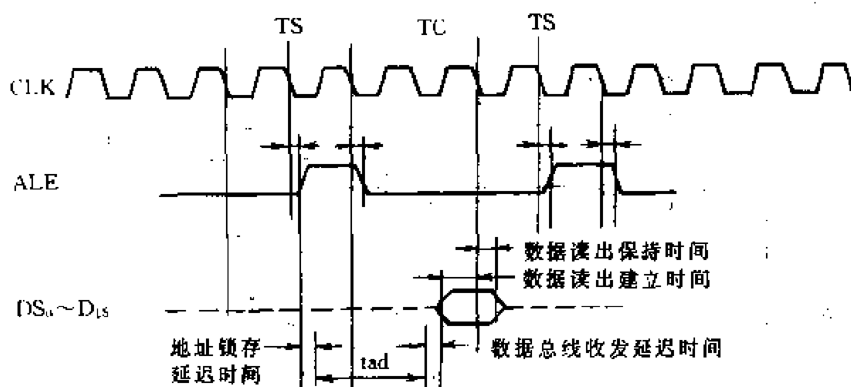


图 6-30 把 ALE 作为地址锁存选通信号时，存储响应的最大富余时间

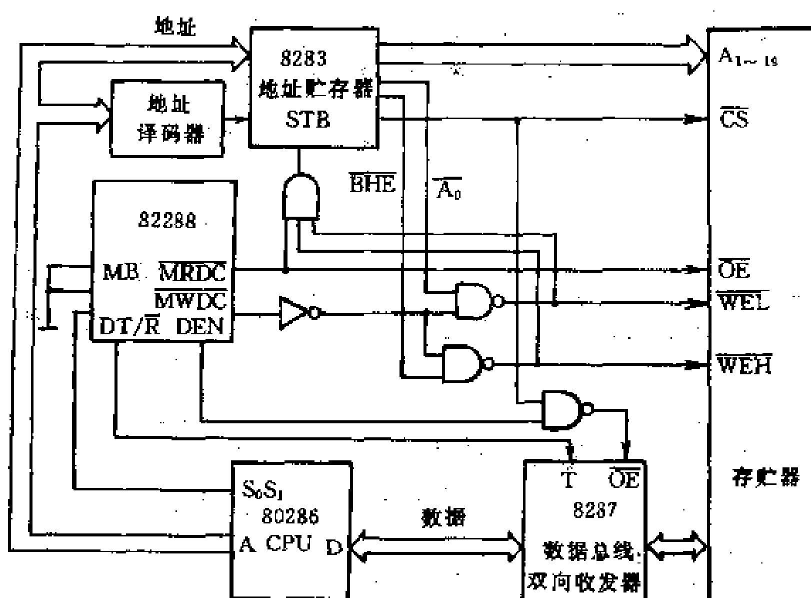


图 6-31 使用特殊选通信号的存储器

16K EPROM 芯片有地址线 14 根即 A_0～A_{13}，把它们分别与地址总线 XA_1～XA_{14}相连，地址信号 A_0 未用，所以系统板上的 ROM 是以字为单位来访问的。这样可在奇偶存储体中，能使用同一个地址信号同时进行操作。至于选择哪一组 ROM，需经译码电路来确定。这样就可以给每一组 ROM 分配确切的地址范围。

(4) 读写存储器(RAM)

表 6-8 是 80286 系统的地址分配情况，由表可见，不管 80286 的 高 4 位地址 A_{20}～A_{23}是全 0(0F0000～0FFFFF)或全 1(FF0000～FFFFFF)，都是指向系统板上的 64KB 的 ROM 区域。与此同时，在系统板上保留的 640KB 也被分配在从 0E0000 或 FE0000 开始的区域。

表 6-8 系统板存储器地址空间的分配

地址空间	配 置	分 配
000000～07FFFF	512KB 系统板	系统板存储器
080000～09FFFF	128KB	I/O 通道存储器，IBM PC/AT 的 128KB 任选扩展存储器

地址空间	配　置	分　配
0A0000～0BFFFF	128KB VRAM	保留给图形显示适配器
0C0000～0DFFFF	128KB I/O 扩充 ROM	保留给 I/O 适配器上的 ROM
0E0000～0EFFFF	在系统板上保留 64KB	副本分配在地址 FE000
0F0000～0FFFFF	系统板上的 64KB ROM	副本分配在地址 FF000
100000～FDFFFF	最大存储量 15MB	I/O 通道存储器,IBM PC/AT 的 512KB 任选扩展存储器
FE0000～FEFFFF	在系统板上保留 64KB	副本分配在地址 0E000
FF0000～FFFFFF	在系统板上的 64KB ROM	副本分配在地址 0F000

80286RAM 电路与 8086 类似。系统 DRAM 存储器由 4 个存储体组成，每个存储体由 9 个 MK41128(128K×1 位)芯片组成,其中有 1 片为奇偶校验位而设置的。MK41128 芯片有 8 位地址线 A_0～A_7,一个写信号$\overline{WE}$,一个列地址选通信号$\overline{CAS}$和 2 个行地址选通信号$\overline{RAS_0}$、$\overline{RAS_1}$。可以把 MK41128 芯片看成由 2 个 64K×1 位组成,各有 1 个行地址寄存器,分别由$\overline{RAS_0}$和$\overline{RAS_1}$控制 2 个 64K×1 位芯片,即用$\overline{RAS_0}$或$\overline{RAS_1}$把 8 位行地址打入到其中一个行地址寄存器。4 个存储体其容量为 512K 字节。需 19 根地址线 A_0～A_{18};同时又可把这 4 个存储体分成两组形成 2 个存储库。RAM_0、RAM_1、RAS、CAS 信号如表 6-9 所示。它由 A_{18}、A_{17}、A_0 地址线和$\overline{BHE}$线组合产生。

表 6-9　RAS/CAS(行/列)选通信号表

	$A_{18}A_{17}$	$\overline{BHE}$　A_0		$A_{18}A_{17}$	$\overline{BHE}$　A_0
	0　0			1　0	
	(RAS$_0$)	0　0		(RAS$_2$)	0　0
	0　1	(CAS$_0$)		1　1	(CAS$_1$)
	(RAS$_1$)			(RAS$_3$)	
	0　0			1　0	
	(RAS$_0$)	0　1		(RAS$_2$)	0　1
	0　1	(CAS$_{0H}$)		1　1	(CAS$_{1H}$)
	(RAS$_1$)			(RAS$_3$)	
RAM_0	0　0		RAM_1	1　0	1　0
	(RAS$_0$)	1　0		(RAS$_2$)	(CAS$_{1L}$)
	0　1	(CAS$_{0L}$)		1　1	
	(RAS$_1$)			(RAS$_3$)	
	0　0			1　0	
	(RAS$_0$)	1　1		(RAS$_2$)	1　1
	0　1	(刷新)		1　1	(刷新)
	(RAS$_1$)			(RAS$_3$)	

A_{18}通过地址译码电路产生 RAM_0 和 RAM_1 的列选通信号$\overline{CAS_0}$和$\overline{CAS_1}$；行选通信号 RAS_0、RAS_1、RAS_2、RAS_3实际上是 A_{18}和 A_{17}译码的 4 种状态。而列地址选通信号$\overline{CAS}$与 A_0、$\overline{BHE}$的组合情况如下：

	A_0	$\overline{BHE}$
	0	1→CAS_{0L}
$A_{18}(RAM_0)=0$	1	0→CAS_{0H}
	0	0→CAS_0
	1	1→刷新

	A_0	$\overline{BHE}$
	0	1→CAS_{1L}
$A_{18}(RAM_1)=1$	1	0→CAS_{1H}
	0	0→CAS_1
	1	1→刷新

而 MK41128 选通信号与系统形成信号连线图如 6-32 所示。

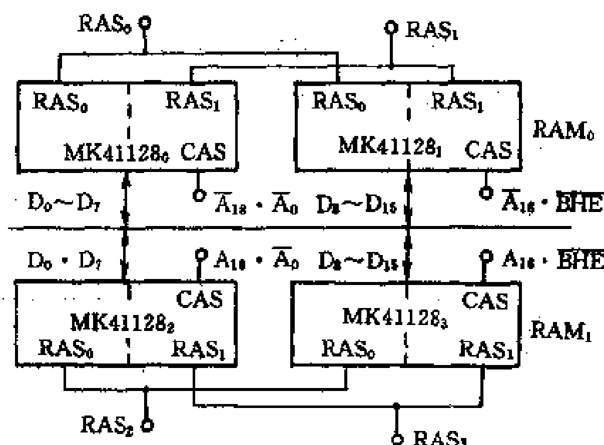


图 6-32 MK41128 选通信号连线示意图

2. 80386 存储系统

(1) 常规存储器接口

如图 6-33 所示，存储器分成 4 个存储体，每个存储体由$\overline{BE}(0\sim3)$来选通，这样可以构成 32 位数据。当$\overline{BE_0}\sim\overline{BE_3}$同时有效时，在一个总线周期里就可完成 32 位数据的存储器读写操作。

(2) 与 ROM 的连接

常规的 TTL 电路或者 PAL 器件可以成为 386 和存储器之间的接口电路，基本的存储器接口框图如图 6-34 所示。总线控制逻辑电路提供了$\overline{READY}\cdot\overline{NA}$、地址锁存、数据收发以及存储器器件控制信号。地址译码电路放在地址锁存之前，它可以提供片选信号、$\overline{BS_{16}}$和总线控制逻辑输入信号，这些常规电路的设计对 ROM、EPROM 和 SRAM 都适合。

图 6-34 中所示的总线控制逻辑相似于 80386 的各种输出($D/\overline{C}$、$W/\overline{R}$和 $M/\overline{IO}$)信号，并提供了用于总线操作类型所需的$\overline{MRDC}$、$\overline{MWTC}$、$\overline{IORC}$、$\overline{IOWC}$、$\overline{INTA}$信号。

地址译码器基本上是在变换地址总线上出现的一个地址为芯片选择信号，地址译码器可以放在地址锁存前或后。

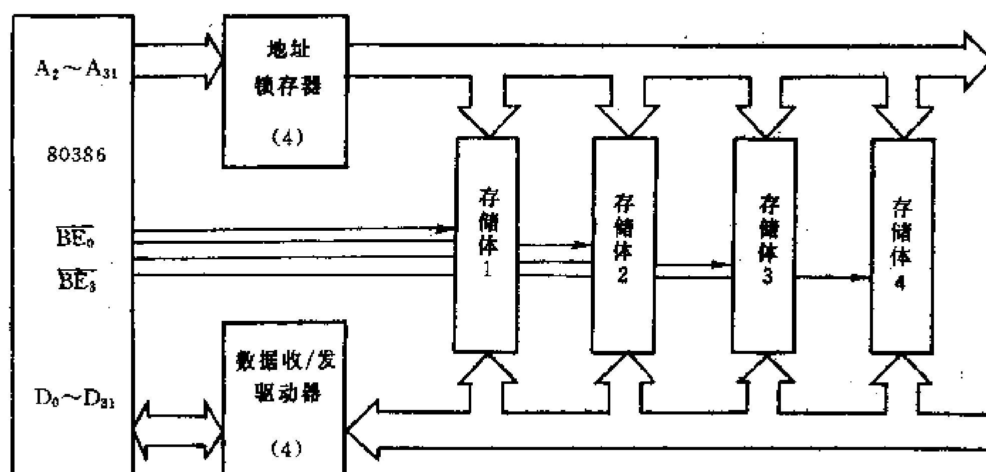


图 6-33 常规存储器接口

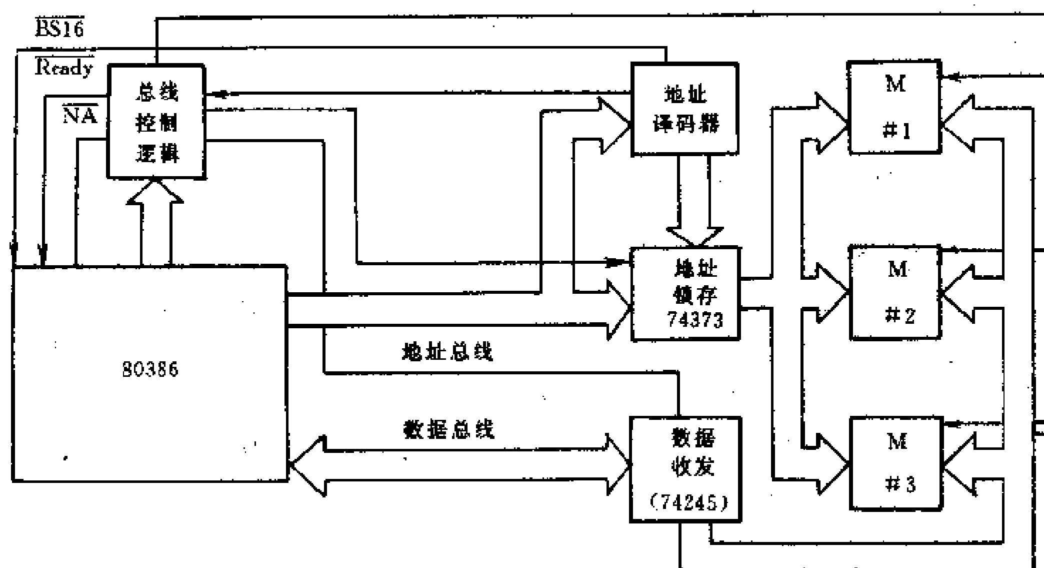


图 6-34 典型的存储器接口框图

地址锁存器像地址译码器，直接连到 386 的地址总线上，当地址译码器获得一个合适的地址，地址锁存器被通知“读入”或锁住这个地址，整个总线周期中维持着该地址信号有效，这样做主要是为了流水线应用，因为下一总线周期的新地址要在当前周期结束前就出现在地址总线上。

数据收发缓冲器放在 386 与存储器之间，可以用 4 个标准的 8 位收发器来构成。它们把 386 和存储器隔离开来，因为有的器件传送数据太慢，如果不隔离，当一个写周期后紧跟一个读周期时，386 可能驱动上一个周期的数据总线，因较慢的存储器还来不及移走它的输出，这样就会造成混乱。

(3)与 RAM 连接

DRAM 不像 SRAM，它需要很短的重新充电的周期以维持内部电荷，否则数据就会丢失，如果同样的存储器被连续调用，访问的时间就会因为重新充电而放慢了。这样，所有的 DRAM 的存储器设计都要保证同样的存储器地址不会被连续调用。一种办法是把存储器分成库，在其他库被访问时，允许这个库重新充电，当我们把相邻的地址放在不同库中时，这种交叉

操作模式就可以避免上述问题出现。

目前有的386计算机使用一种存储器叫静态-列存储器，这种方案是把系统存储器分成相同的部分，称为页，存储器访问同一页的速度很快，而访问不同的页的存储器较慢，这种分页技术可以用静态-列存储器来实现。

每一个存储器库区分为行和列的矩阵，然后特殊的"单元"被行和列地址所访问，在静态-列存储器中行地址选通信号(RAS)和列地址选通信号(CAS)锁存这些地址，行和列地址是9位的，一行选址 2^9 得到512"单元"，每一列也有512个"单元"。386计算机存储单元典型的有32位(32×512行=2KB为一页)在一页里先进行一次行选后，只进行列选就能存取1MB的DRAM存储器这种页模式访问需要0～2个等待状态。

常用的有82C302页式/交叉内存控制器来控制DRAM。内存配置可以是一个存储体(无交叉存储)，可以是2个或4个交叉存储体的存储库，82C302能实现页或页/交叉操作模式。下面介绍另一种存储器即高速缓冲存储器。

高速缓冲存储器是由小容量的SRAM和高速缓存控制器组成。它的功能是把386CPU要用到的数据由DRAM复制到SRAM中，而由SRAM向386CPU直接提供它所要用到的大多数数据，实现零等待状态。因DRAM容量大称为主存储器，它和高速缓冲存储器一起构成了动态存储器系统。高速缓冲控制器根据386CPU所需数据来决定，主存储器中哪个数据块需移出，需移到SRAM中，或在SRAM中哪个数据块要传送给386CPU等。这样就用小容量SRAM来模拟大容量的DRAM，用少量的花费却得到了相当于SRAM的性能的存储器。命中率高的高速缓冲存储器系统的存取速度，接近SRAM系统，有的命中率可高达98%，它根据从一个地址预测下一个存储器地址很可能接近上一地址单元的概念，将主存储器分成块，一块大约2～32字节。高速缓存控制器取一个存储块而不是取一个地址。这样的块可以包含处在所需字节前后的数据。一般块越大，命中率就越高，但会使高速缓存中的块变少，因而从主存储器来的块重写操作就变得频繁了，每一个块中的字离所需字也越远，所以要大小适宜。

直接变换的高速缓冲存储器

如图6-35所示为64KB高速缓冲存储器与16MB的主存接口示意图，32位物理地址被分为3个字段。其一是选择字段通过片内选择逻辑决定到高速缓冲存储器访问还是到主存访问。后面8位标志再加上16位变址用来决定16MB的主存地址($2^{24}=16MB$)，它可以访问16MB的DRAM。而16位变址字段用来对SRAM寻址($2^{16}=64K$)，所以16位变址可遍访64KB高速缓冲存储器。64K×8数据SRAM对32位数据总线来说，也可组成16K×32位SRAM，在16M×8位DRAM中，能有256个16K×32位这样大的数据组，该256也是标志字段 $2^8=256$；16K×32位SRAM是用 $A_2 \sim A_{15}$ 14位地址线来寻址，低两位可由 $BE_3 \sim BE_0$ 字节选择来完成。若将DRAM数据分为4字节一块，每一块只含一个地址单元。当CPU需要数据时，发出一个24位地址信号，高速缓存控制器找出CPU所需的变址字段中地址，该地址中的数据是否为CPU所需数据呢？这还要用地址中的标志字段内容与在这之前将变址地址中的数据存入SRAM时，其存入高速缓存器标志存储器中所存地址内容进行比较。只做一次性比较，若匹配，CPU将从SRAM中读出数据；若不匹配就必须执行对DRAM的访问，使CPU获得所需的数据。数据不仅被送入CPU，而且还被存入SRAM中。直接变换高速缓冲存储器的缺点是，主存DRAM组间进行频繁调用时，高速缓存控制器必须做很多切换工作。

两路相联高速缓冲器

为了克服以上缺点，采用两路相联高速缓冲存储器，如图6-36所示。此类型使用了两个直

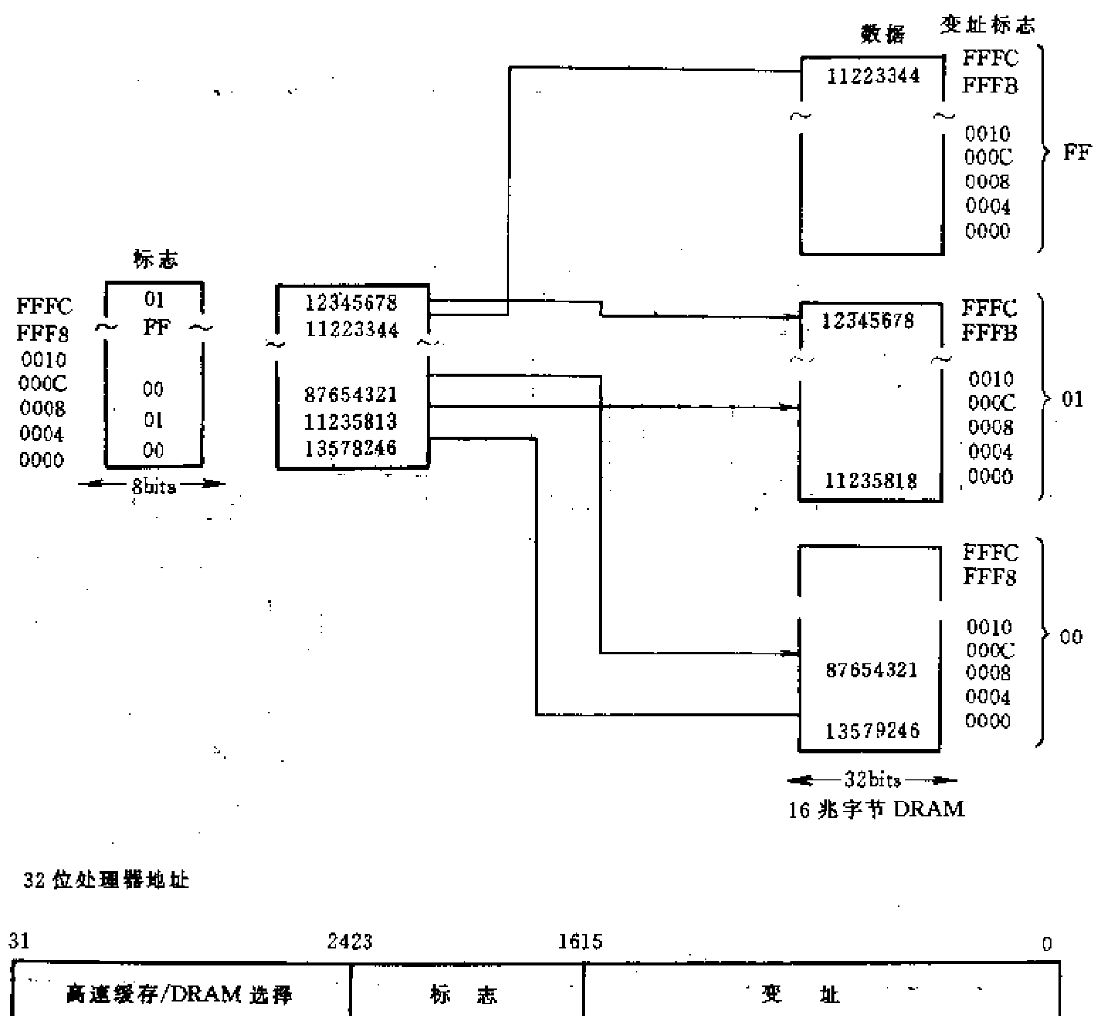


图 6-35 直接变换的高速缓冲存储器机构

接变换块，对它们一起操作，如像是两个直接变换的高速缓冲存储器；对任一给定的变址值，两个块单元皆响应。这种方案增加了命中率和有效性。因为它对同样变址块用了两个标志地址。

在高档 386 机中一般使用 82385 作为高速缓存控制器，含有控制逻辑和标志存储器。

3. 80486 存储系统

80486 内含一个 8KB 的高速缓存，它为 4 路成组联想式。一个高速缓冲存储行由存储器的 16 个字节邻接块组成。486CPU 能够用一个时钟将指令或数据从其内部高速缓存传送到指令预取队列缓冲器或执行部件中。当出现高速缓存命令时，许多指令只需一个时钟就可完成。

高速缓存的命中率是随应用的不同而异，但它还是趋向于 90%以上。就是说高速缓存还会出现未命中的情况。这些未命中同那些不能高速缓存(例 I/O 等)一起，就会要求 486 微处理器硬件接口性能设计要特别高。为了获得高性能的 486CPU 的系统设计，了解与硬件设计有关的折中方案是很重要的。

存储器系统的设计是决定系统性能的一个主要因素。在 486CPU 上的 8KB 高速缓存满足处理器的许多请求的时候，多任务处理和多重处理软件的一些要求有时会违反与这种容量的高速缓存相关的局部性原则。为保持高性能，拥有一种高速的存储器设计是很重要的。这可以通过优化主存(DRAM)的设计或者增加一个二级高速缓存来实现。

(1)DRAM 主存储器

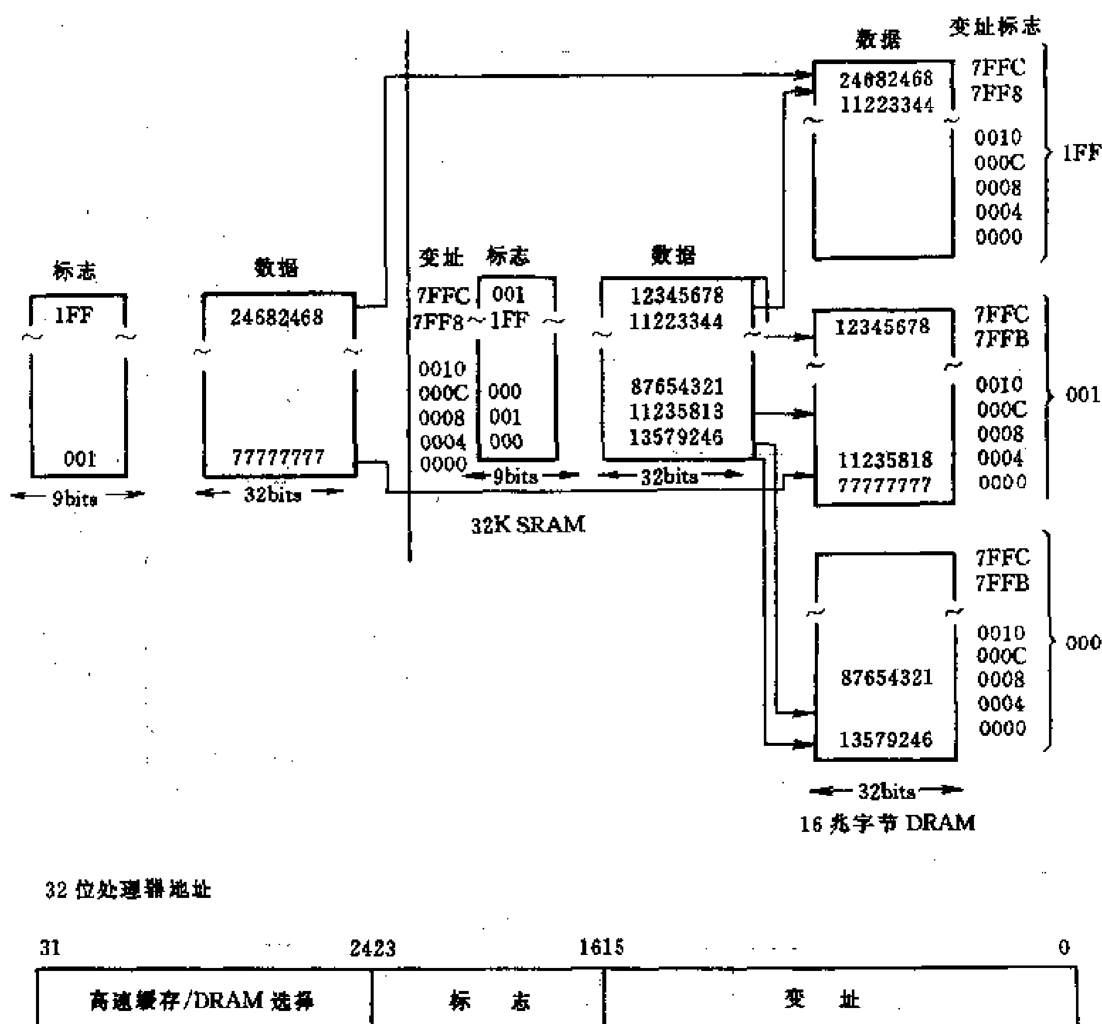


图 6-36 64K 两路相连高缓冲存储器机构

大多数系统的主存都用DRAM 构成，因为它集成度高，成本低。但 DRAM 需要刷新，它的存取时间大于微处理器读写时间。完成对 DRAM 主存访问时，CPU 被迫要等待 DRAM 存取的完成。为了减少等待状态时钟周期数量，可以优化 486CPU DRAM 的设计和突发式传送，存储器交叉和写登记(WRITE-POSTING)等。

486CPU 采用突发式传送，来加速高速缓冲行填充这样的多周期数据传送。突发式传送是利用"一知道后续传送地址就发出第一个地址"。虽然第一个传送需要完整的存储器访问时间，但随后的传送可以提前开始并与前面的访问重叠。突发式传送提供的最大性能优势是在高速缓冲行的填充期间。如果存储器系统的速度足够高时，突发式传送用 5 个时钟周期就可以进行一次高速缓存行的填充，两个时钟周期用于第一个传送，随后的三个传送各用一个时钟周期。在没有突发式传送的系统，进行同样的传送任务最少需要 8 个时钟周期。所有运行基准检查程序系统都是利用 486CPU 的突发式传送能力。

存储器交叉是一种改进 DRAM 设计便于实现突发式传送的方法。虽然实现存储交叉有若干种方法，但用于突发式传送的一种简单方法是设计两个 32 位宽的 DRAM 存储体。数据在两个库之间交替地存储。偶数 DWORD($A_2=0$)放入一个存储库中，而奇数 DWORD($A_2=1$)放入另一库中。突发式传送以交替的顺序在两存储库之间进行访问。相邻的周期访问时间可以重叠，以便能有效地对单时钟 DWORD 进行访问。在两个 DRAM 存储库之间采用多路转换器，

以实现交替访问数据和地址总线，避免冲突。

存储器分页是用于提高DRAM存储器性能的另一种方法。将存储器分成若干页面(典型的每个页面为2K字节)，可以更快地对同一页面进行连续的存储器访问。DRAM的访问在正常情况下，需要一个列地址和一个行地址。分页方式的DRAM从最初访问时就"记住"了行地址，只要是对同一页面进行访问，仅需一个列地址就行，因而减少了存储器访问所需的时间。

写登记允许设计者优化在486CPU系统中所遇到的高写入百分比的存储器设计，由于芯片上有8KB的高速缓存，所以所有的存储器总线周期70%以上都是写入。通过写登记，系统在存储器实际接收数据之前CPU就已完成写周期的信号发送。CPU可以提早开始执行下一条指令，减少写周期中CPU等待状态的数量。登记有可变的级别是可能的。486CPU可将多达4个写入周期登记到外部存储器上。外存的单写入登记缓冲区可以提高性能，而又不会使存储器设计增加更多复杂性。

在有二级高速缓存系统中，写入缓冲区时，存储器控制必须保证在写入完成前，不能对被登记的写入地址进行读出。最简单的方法是延迟所有的DRAM读出周期，直到被登记的写入完成为止。这种方法只延迟了高速缓冲存未命中的读出周期，而命中周期能继续执行，从而提高了系统的性能。

(2) 二级高速缓存

二级高速缓存的价格性能比适于SRAM与速度较慢的DRAM组合在一起以降低整个存储器的存取时间。二级高速缓存可避免一种相对无效的DRAM存储器设计。通过采用二级高速缓存连接速度较慢的但价廉的DRAM存储器系统可以降低系统的成本。

二级高速缓存可以提高性能，但提高多少取决于DRAM的实现方式。采用快速DRAM设计，对从二级高速缓存中所获得的性能提高很小。

在486CPU系统中对二级高速缓存的需要通常是视应用而定的。对于许多应用来讲，8K字节的片内高速缓存就足够了。如果软件或硬件的配置需要一个二级高速缓存时，则有若干可供考虑的选择方案。在单486处理器系统中存储器总线的利用率通常为50%至70%之间。所以从单CPU系统中使用二级高速缓存所得到的好处与降低总线利用率相反，但缩短了存储器的平均时间。485Turbocache模块对基于单486CPU的系统是一种任选的二级高速缓存。485Turbocache模块是一种后备通写式高速缓存，有64K字节和128K字节两种方案可供选用。这种后备式的特点是允许Turbocache模块成为任选的，允许用户根据使用最频繁的软件来决定对高速缓存的需要。当出现高速缓存命中时，后备高速缓存监控CPU的信号和对存储器进行干预。通写的意思是对高速缓存的所有写入均立即传送到主存中去。

在使用多486CPU的系统中总线的利用可能会成为一个问题，回写式高速缓存除了缩短存储器的平均存储时间之外。还会降低总线的利用率。在回写式高速缓存情况下，各次写入都被记入高速缓存。当高速缓存的单元是空的或被取代，或者是另一个处理器正在从主存中读出该单元和需要最近的数据时，高速缓存仅将写入传送到主存。回写式高速缓存实现起来比通写式高速缓存复杂，因为增加了对高速缓存/主存一致性的要求。除了总线利用率这个问题外，通写式高速缓存可以充分地缩短存储器的平均存取时间，并能满足大部分系统的需要。

思考题与习题

1. 内存储器性能的主要指标是哪几个?

2. 半导体存储器(RAM 和 EPROM)与 CPU 的连接应注意哪些方面?

3. 某 SRAM 的一单元中存放有一个数据如(5AH),CPU 将它取走后,该单元的内容是什么?

4. 若某微机有 16 条地址线,现用 SRAM 2114(1K×4)存储芯片组成存储系统,问采用线选译码时,系统的存储容量最大为多少? 此时需要多少个 2114 存储器芯片?

5. EPROM 存储器芯片还没有写入信息时,各个单元的内容是什么?

6. 下列 RAM 各需要多少个地址输入端?

512×4	1K×8	1K×4	1K×1
4K×1	16K×1	64K×1	256×1

7. 把由 3000 条指令和数据字组成的同样的程序和数据存储在以下 2 种类型的存储器中:只读存储器、半导体随机存取存储器。若发生掉电现象,试问那种类型的存储器仍然含有上述程序和数据?

8. 对下列微型机系统中的 DRAM 存储器进行刷新,各需几次才能刷新完毕?所需刷新地址计数器各有几位触发器组成?

(1)由 4K×1 DRAM 芯片组成 16K×8 位存储器

(2)由 4K×1 DRAM 芯片组成 64K×8 位存储器

(3)由 16K×1 DRAM 芯片组成 64K×8 位存储器

9. 下列 ROM 各需要多少个输入端? 多少个输出端?

(1)16×4 位 ROM　　(2)32×8 位 ROM

(3)256×4 位 ROM　(4)512×8 位 ROM

10. 已知某微机控制系统中的 RAM 容量为 4K×8 位,首地址为 4800H,求其最后一个单元的地址。

11. 某微机系统中内存的首地址为 3000H,末地址为 63FFH,求其内存容量。

12. 某单板机中 ROM 为 6K,最后一个单元的地址为 9BFFH,RAM 为 3K、已知其地址为连续的,且 ROM 在前,RAM 在后,求该存储器的首地址和末地址。

13. 设有一个具有 14 位地址和 8 位数据的存储器,问:

(1)该存储器能存储多少字节的信息?

(2)如果存储器由 8K×4 位 RAM 芯片组成,需要多少片?

(3)需要地址多少位作芯片选择?

14. 对 DRAM 的刷新方式一般有哪几种?

15. 用 16K×1 位的 DRAM 芯片组成 64K×8 位的存储器,要求:

(1)画出该存储器组成的逻辑框图。

(2)设存储器读、写周期均为 0.5μs,CPU 在 1μs 内至少要访存一次。试问采用哪种刷新方式比较合理? 两次刷新的最大时间间隔是多少?

16. 有一个 16K×16 的存储器,由 1K×4 位的 DRAM 芯片构成(芯片内是 64×64 结构),问:

(1)总共需要多少 RAM 芯片?

(2)存储器的组成框图。

(3)采用异步刷新方式,如单元刷新间隔不超过 2ms,则刷新信号周期是多少?

(4)如采用集中刷新方式,存储器刷新一遍最少要用多少读/写周期?

17. 某微机系统中,用 1 片 EPROM2732,它与 CPU 的连接如图 6-37 所示,求此芯片的存储器容量及地址空间的范围。

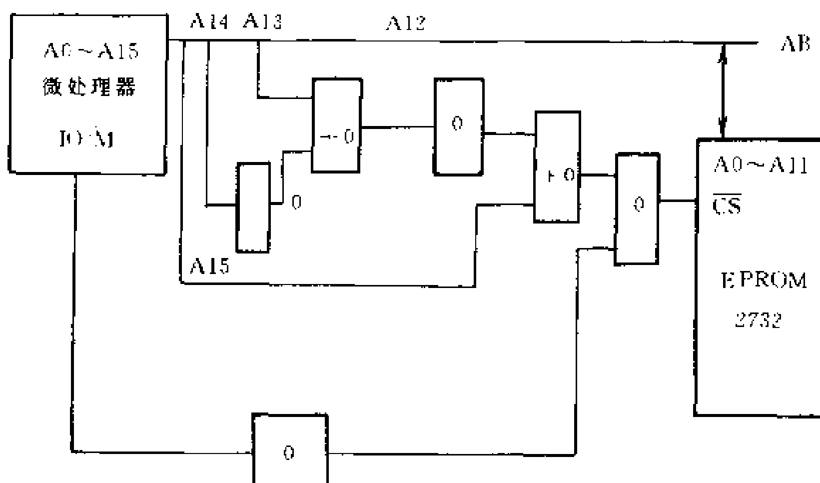


图 6-37

18. 某微机系统中,用 2 片 EPROM27128(16K×8)和 2 片 SRAM6264(8K×8)以及一个 3-8 译码器(74LS138)来组成存储系统,各集成芯片的主要信号如图 6-38 所示,要求起始地址为 000H,画出系统连接图,并写出每一存储芯片的地址空间范围。

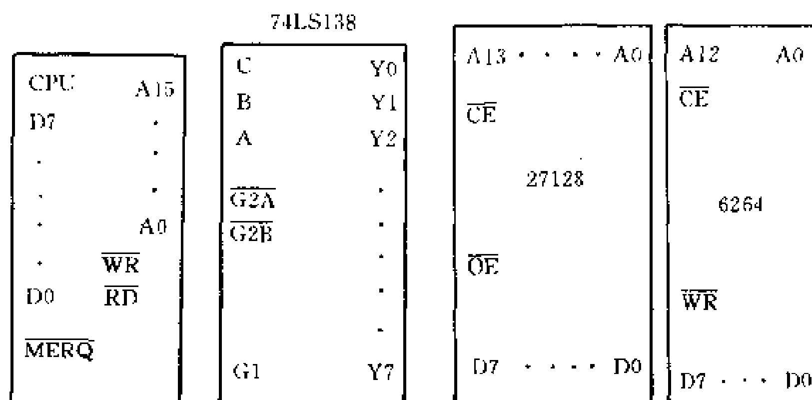


图 6-38

第七章　输入和输出系统

第一节　概　述

输入/输出(I/O)是指主机与外界交换信息——即通信。主机与外界的通信是通过输入/输出设备进行的。I/O 系统包括 I/O 接口、I/O 设备、I/O 管理部件和与 I/O 有关的软件。一个微机系统的综合处理能力、系统的可靠性、兼容性、性能价格比、甚至在某个场合能否使用,都和 I/O 系统有着密切的关系。输入和输出系统是计算机系统的重要组成部分之一,可以这样说,任何一台高性能计算机,如果没有高质量的输入输出系统与之配合工作,计算机的高性能便无法发挥出来。

一、I/O 与接口电路

输入输出系统应包含输入输出设备以及它们与计算机之间的接口。这是因为一般的输入输出设备都是机械的或机电相结合的产物,比如常规的外设有键盘、CRT 显示器、打印机、磁盘、磁带、纸带阅读机、光电输入机等,它们相对于高速的中央处理器来说,速度要慢得多。此外,因外设的信号形式、数据格式各异,为了要把外设与 CPU 连接起来,必需有接口部件,以完成它们之间的速度匹配、信号匹配和完成某些控制功能。对于小型、中型以上的计算机,常常是在设计计算机主机时就连接口部件一起设计出来,因此通常它们专用于某个计算机系统。而对于微型计算机来说,设计微处理器 CPU 时,并不设计它与外设之间的接口部件,而是将输入输出设备的接口设计成完全独立的部件,使它们成为通用的接口芯片,通过它们可以将各种类型的外设与 CPU 连接起来构成完整的微型计算机系统。

二、CPU 与外设间交换的信息

I/O 外设与 CPU 之间交换的信息有数据、状态及控制信号。如图 7-1 所示。

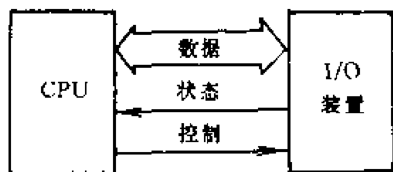


图 7-1　CPU 与外设间传送的信息

1. 数据信号

数据信号又可分为数字量、模拟量和开关量三种。

数字量:由键盘、CRT、打印机及磁盘等 I/O 外设与 CPU 交换的信息,是以二进制形式表示的数或以 ASCII 码表示的数或字符。

模拟量:当计算机用于控制系统中时,大量的现场信息经过传感器把非电量(如温度、压力、流量、位移等)转换成电量,并经放大处理得到模拟量电压或电流。这些模拟量必需先经过 A/D 转换器转换后才能输入计算机;计算机输出的控制信号也必须先经过 D/A 转换器,把数字量转换成模拟量才能去控制执行机构。

开关量:这是一些两个状态的量。如开关的断开和闭合,机器的运转与停止,阀门的打开与关闭等。这些开关量,通常要经过相应的电平转换才能与计算机连接。这些开关量只要用一

位二进数即可表示，故对字长为 8 位(或 16 位)的计算机，一次可输入或输出 8 位(或 16 位)开关量。

数据的传输可采用并行传送(n 位同时传送)和串行传送(一位一位地传送)两种形式。在这一章中，我们只讨论并行传送。

2. 状态信号

状态信号是指在 CPU 与外设之间交换数据时的联络信息。CPU 通过对外设状态信号的读取，可得知其工作状态。如输入设备的数据是否准备好，输出设备是否空闲，若输出设备正在输出信息，则用 BUSY 信号通知 CPU 以便暂停送数。因此，状态信号是 CPU 与 I/O 外设正确进行数据交换的重要条件。

3. 控制信号

控制信号是用来设置 I/O 外设(包括 I/O 接口)的工作模式的。

I/O 接口的基本作用是提供数据缓冲、完成信息格式的变换、管理数据传送、实现电气特性的匹配以及进行地址译码或选择设备等。

三、接口电路的功能

常用的 I/O 接口具有如下功能：

(1) 能提供计算机与外设间有关信息格式的相容性变换，如正负逻辑转换，串并行间的转换等。

(2) 能对传送数据提供缓冲，用以消除计算机与外设在"定时"或数据处理速度上的差异。

(3) 能提供有关数据传送的协调状态，如设备"准备好"，数据缓冲器"空"或"满"。

(4) 能提供有关电气特性的适配，尤其是用 MOS 工艺制造的微处理机，由于它的输出电流、电平及输出能力与外设要相适配，其间必须要有缓冲电路。

(5) 能对外设进行中断管理，如建立中断请求，进行中断排队，提供中断识别码等。

(6) 能进行地址译码，对某些输入/输出设备，还需要有对命令进行译码的功能。

(7) 能提供时序控制，如有自已的时钟发生器以满足计算机和各种外设在时序控制上的需要。

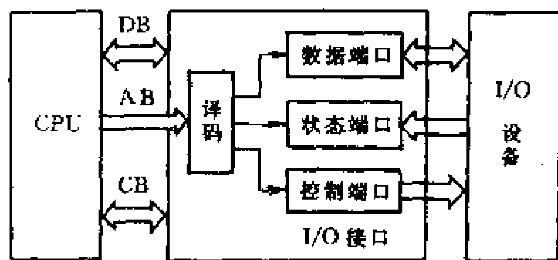


图 7-2 一个典型的 I/O 接口

图 7-2 是一个典型的 I/O 接口，其中既有数据端口，又有状态及控制端口。每个 I/O 端口对应一个 I/O 地址。从硬件上看，端口可以理解为寄存器。CPU 用 I/O 指令对其直接访问。图 7-2 中的接口电路通常占有三个 I/O 地址(亦称端口地址)，分别对应数据端口、状态端口和控制端口。其中数据端口可以是双向的，而状态端口只有输入操作，控制端口只作输出操作。有时后两个端口合用一个端口地址，再用 I/O 读(IOR)或 I/O 写(IOW)信号来分别选择访问。

四、I/O 讨论的问题

I/O 主要讨论两个问题：

1. 主机与那个设备进行 I/O？即 I/O 的编址方式。

2. 主机与外设之间怎样实现 I/O？即 I/O 传送的控制方式。

至于各类通用接口芯片的功能，结构及其使用方法在第九章讨论。而 I/O 设备本身的结构及其工作原理限于篇幅，可参考相关资料。

第二节 I/O 端口的编址方法

每个接口部件都包含一组寄存器，如图 7-3 所示。CPU 和外设进行数据传输时，各类信息在接口中存入不同的寄存器，一般称这些寄存器为 I/O 端口，每个端口有一个端口地址。

用于对来自主机的数据或者送往主机的数据起缓冲作用的端口叫数据端口。

用来存放外设或者接口部件本身状态的端口，称为状态端口。CPU 通过对状态端口的访问可以检测外设和接口部件当前的状态。

用来存放 CPU 发出的命令，以便控制接口和设备动作的端口叫命令端口。

可以说，计算机主机和外部设备之间都是通过接口部件的 I/O 端口来沟通的，所以，CPU 就一定有和 I/O 端口打交道的办法。

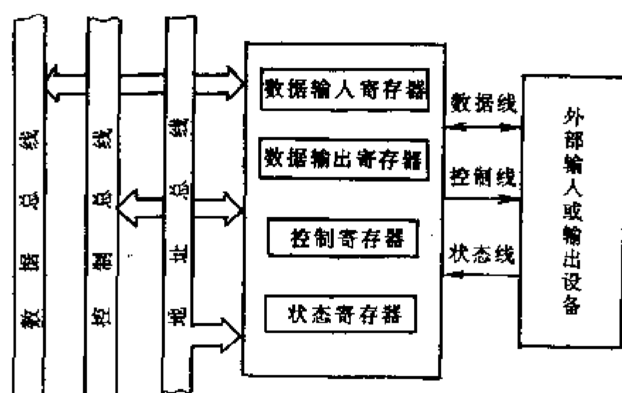


图 7-3 外部通过接口和系统的连接

在微型计算机中常用两种 I/O 端口编址方式，介绍如下。

一、I/O 端口地址与内存单元地址统一编址方式

这种编址方式又称为存储器映射编址方式。在这种编址方式中，将 I/O 端口地址和内存地址统一安排在内存的地址空间中。即把内存的一部分地址分配给 I/O 端口，由端口来占用这些地址。用于 I/O 端口的内存地址，存储器不能再使用。这样，计算机系统的内存空间一部分留做端口地址来使用，而剩下的内存空间可做内存使用。

I/O 端口与内存统一编址的方法占用了部分内存空间，将 I/O 端口看作是内存单元。因此，原则上说，用于内存的指令都可以用于外设，这给使用者提供了极大的方便。但由于 I/O 端口占用内存地址，就相对减少了内存可用范围 。而且，从指令上不易区分是访内的指令还是用于输入输出的指令。这种编址方式在 68 系列和 65 系列的微型机中得到广泛地应用。

二、I/O 端口与内存独立编址

在这种编址方法中，内存地址空间和 I/O 端口地址是相对独立的。例如，在 8086(8088) CPU 中，内存地址是连续的 1M 字节，从 00000H～FFFFFH。而 I/O 端口的地址范围从 0000H ～FFFFH。它们相互独立，互不影响。这是由于 CPU 在访问内存和外设时，使用了不同的控制信号来加以区分。8086CPU 的 M/$\overline{IO}$信号为 1 时，表示地址总线上有一个内存地址；当它为 0 时，则表示地址总线上的地址是一个有效的外设地址。而 8086CPU 为 IO/$\overline{M}$，请读者注意。

内存与端口独立编址，各自有自己的寻址空间。用于内存和用于 I/O 端口的指令是不一样的，很容易辨认。但用于 I/O 端口的指令功能比较弱，一些操作必须由外设首先输入到 CPU 的寄存器后才能进行。这种编址方式在 Z80 系列及 INTEL80 系列微机中得到广泛采用。

第三节　输入/输出控制方式

在各种微型计算机系统中，可采用的输入/输出控制方式有直接程序控制方式、中断控制方式、直接存储器存取方式(简称 DMA 方式)和输入/输出处理器方式(简称 IOP 方式) 。在 8086/8088 系统中，上述四种方式均可使用，为了较好地发挥 CPU 的效率，故较多地采用后三种方式。

一、直接程序控制的 I/O 方式

直接程序控制 I/O 方式传送是指直接在程序控制下进行 I/O 信息传送，又分为无条件传送方式和条件传送方式。

1. 无条件传送方式

如果程序员能够确信一个外设已经准备就绪，那就不必查询外设而进行信息传输，这称为无条件传送方式。

在无条件传送方式下，程序设计较简单。不过名为无条件传送，实际上对传送是有条件的，那就是传送不能太频繁，以保证每一次传送时，外设处于就绪状态。所以无条件传送方式用得较少，只用在对一些简单外设的操作，如开关，七段显示管等。

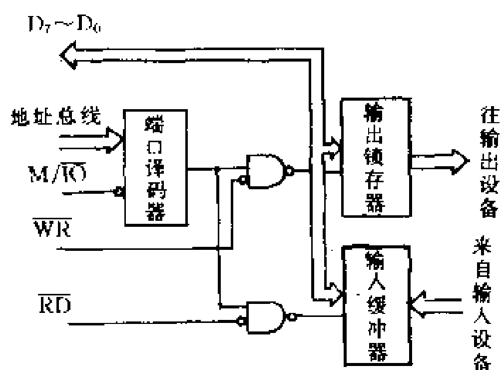


图 7-4　无条件传送方式的工作原理

由于简单外设作为输入设备时，输入数据保持时间相对于 CPU 的处理时间要长得多，所以可直接使用三态缓冲器和数据总线相连，如图7-4所示。当执行输入的指令时，读信号$\overline{RD}$有效，选择信号 M/$\overline{IO}$处于低电平，因而三态缓冲器被接通，使其中早已准备好的输入数据送到数据总线上，再到达 CPU。可见要求 CPU 在执行输入指令时，外设的数据是准备好的，即已经存在三态缓冲器中，否则出错。

简单外设作为输出设备时，一般都需要锁存器，也就是说，要求 CPU 送出的数据在接口电路的输出端保持一段时间。其原因仍然是由于外设的速度比较慢，所以，要求 CPU 送到接口的

数据能保持和外设动作相适应的时间。如图 7-4 所示，CPU 执行输出指令时，M/$\overline{\text{IO}}$和$\overline{\text{WR}}$信号有效，于是，接口中的输出锁存器被选中，CPU 输出的信息经过数据总线送入输出锁存器中，输出锁存器保持这个数据，直到外设取走。显然，这里要求 CPU 在执行输出指令时，确信所选中的输出锁存器是空的。

2. 条件传送方式

条件传送也称为查询传送。用条件传送方式时，CPU 通过执行程序不断读取并测试外设的状态，如果外设数据端口处于准备好状态(输入设备)或者空闲状态(输出设备)，则 CPU 执行输入指令或输出指令与外设交换信息。为此，接口电路除了有传送数据的端口以外，还要有传送状态的端口。对于输入过程来说，当外设将数据准备好时，则使接口的状态端口中的"准备好"标志位置成有效；对于输出过程来说，外设取走一个数据后，接口便将状态端口中的对应标志位置成有效，表示当前输出数据端口已经处于"空闲"状态，可以接收下一个数据。

可见，对于条件传送来说，一个数据传送过程由 3 个环节组成：

■CPU 从接口中读取状态字。

■CPU 检测状态字的对应位是否满足"就绪"条件，如果不满足，则回到前一步读取状态字。

■如状态字表明外设已处于"就绪"状态，则传送数据。

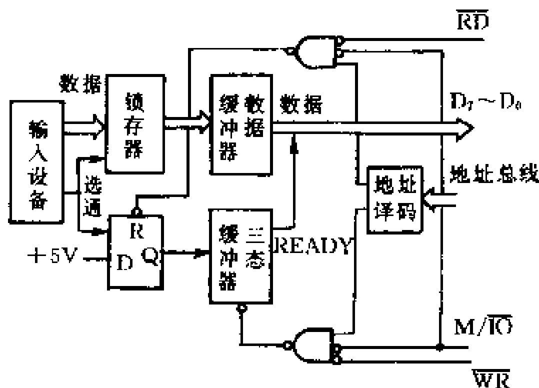


图 7-5 查询式输入的接口电路

图 7-5 表明了用查询方式进行输入的接口电路。输入设备在数据准备好以后便往接口发一个选通信号。这个选通信号有两个作用，一方面将外设的数据送到接口的锁存中，另一方面使接口中的一个 D 触发器置 1，从而使接口中三态缓冲器的 READY 位置 1。数据信息和状态信息从不同的端口经过数据总线送到 CPU。按数据传送过程的 3 个步骤，CPU 从外设输入数据时先读取状态字，检查状态字是否表明数据准备就绪，即数据是否已送入输入数据端口，如准备就绪，则执行输入指令读取数据，且使状态位清 0，这样，便开始下一个数据传输过程。

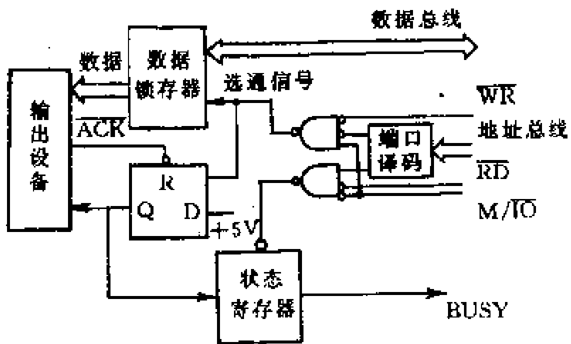


图 7-6 查询式输出的接口电路

图 7-6 表明了用查询方式进行输出的接口电路。当 CPU 要往一个外设输出数据时，先读取接口中的状态字，如果状态字表明外设有空(或"不忙")，这说明可以往外设输出数据，此时 CPU 才执行输出指令，否则 CPU 必须等待。

CPU 执行输出指令时，由选择信号 M/$\overline{\text{IO}}$和写信号 $\overline{\text{WR}}$产生的选通信号将数据总线上的数据送入接口锁存器，同时使 D 触发器置 1，D 触发器的输出信号一方面为外设提供一个联络信号，告诉外设接口中已有数据可供提取；另一方面，D 触发器的输出信号使状态寄存器的对应标志位置 1(有些设备用"忙"标志状态，有些设备用"空"标志状态，两者有效电平正好

相反)告诉CPU,当前外设处于忙状态,从而阻止CPU输出新的数据。

当输出设备从接口中取走数据后,通常会送一个回答信号$\overline{ACK}$,$\overline{ACK}$使接口中的一个D触发器置0;从而使状态寄存器中的对应标志位置0。这样就可以开始下一个输出过程。

归纳起来,查询式输入/输出一般是通过下列过程来实现的:程序先对接口进行连续的检测。当检测到的状态表明接口中已有数据准备输入到CPU或者接口准备好从CPU接收数据时,就可以进行输入/输出操作。

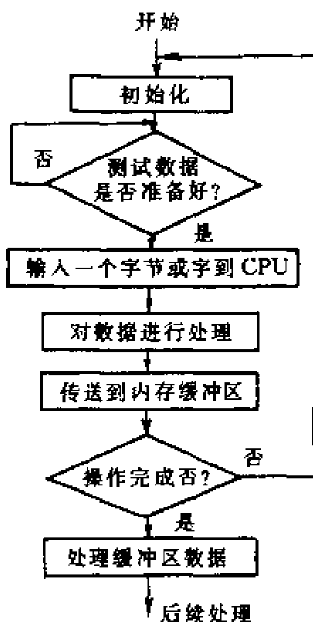


图7-7 查询式输入过程的流程图

图7-7是一个查询式操作的流程图。图中假定要输入1个字节串或者1个字串,每个字节或者字被送到CPU以后进行一定的处理,然后送到内存缓冲区,当所有的数据都输入完毕并送到缓冲区中后,再对缓冲区中的数据进行处理。

下面举一个使用查询式输入输出方式的实例。

假设从终端往缓冲区输入一个字符行。当遇到回车符(0DH)或者字符行超过80个字符时,输入便结束,并自动加上一个换行符(0AH),如果在输入的81个字符中未见到回车符,则在终端上输出信息"BUFFER OVERFLOW"。

因为终端往CPU输入的是ASCII码,而ASCII码采用的是7位二进制表示。所以,用第7位即最高位作为终端往CPU传送时的校验位,这里假定采用偶校验,如果校验出错,输出错误信息,如果校验没有错误,则先清除校验位,再传送到缓冲区。

假定接口的数据输入端口地址为0052H,数据输出端口地址为0054H,状态端口地址为0056H,并且设定如果寄存器中第一位为1,则表示输入缓冲器中已经有一个字节准备好,可以进行输入。此外,还假定状态寄存器的第0位为1,则表示输出缓冲器已经腾空,因而CPU可以往终端输出数据。当然两个假设都是有条件的,条件就是在设计接口部件时,使得状态寄存器的第一位在接口从设备输入一个字节时,便自动置1,而当CPU从接口读取一个字节时,便自动置0;相类似的,当CPU往接口输出一个字节时,状态寄存器的第0位自动置0。而当一个字节从接口输出到设备时,则自动置1。具体程序如下:

```
          DATA_SEG SEGMENT
          MESSAGE DB 'BUFFER OVERFLOW',0DH,0AH
            ⋮
          DATA_SEG ENDS
          COM_SEG SEGMENT
          BUFFER DB 82 DUP(?)          ;接收缓冲区
          COUNT DB ?                   ;计数器
          COM_SEG ENDS
          CODE SEGMENT
          ASSUME DS:DATA_SEG,ES:COM_SEG,CS:CODE
   START: MOV   AX,DATA_SEG            ;对DS作初始化
          MOV   DS,AX
```

```
            MOV   AX,COM_SEG                    ;对 ES 作初始化
            MOV   ES,AX
            MOV   DI,OFFSET BUFFER              ;计数器指向缓冲首址
            MOV   COUNT ,DI
            MOV   CX,81                         ;字符行长度
            CLD                                 ;清方向标志
NEXT-IN: IN   AL,58H                            ;读入状态
              TEST   AL,02H                     ;测状态寄存器第一位
              JZ   NEXT-IN                      ;未准备好,则等待,再测
              IN   AL,52H                       ;准备好则输入字符
              OR   AL,0                         ;校验
              JPE   NO_ERROR                    ;检验正确,则转 NO_ERROR 程序
              JMP   ERROR                       ;检验出错,则转 ERROR 程序
NO_ERROR: AND   AL,7FH                          ;清除校验位
              STOSB                             ;将字符送入缓冲区
              CMP   AL,0DH                      ;是否为回车符?
          LOOPNE   NEXT-IN                      ;不是回车,则再输入
              JNE   OVERFLOW                    ;不是回车且溢出,则转 OVERFLOW
          MOV   AL,0AH                          ;加一个换行符
              STOSB                             ;存入缓冲区
              SUB   DI,COUNT                    ;计算输入字符数
          MOV   COUNT,DI                        ;COUNT 中为输入字符数
OVERFLOW:MOV   SI,OFFSET MESSAGE;SI 中为字符串首址
              MOV   CX,17                       ;字符数
NEXT-OUT:IN   AL,58H                            ;读状态寄存器
              TEST   AL,01H                     ;测状态寄存器第 0 位
              JZ   NEXT-OUT                     ;如果没有就绪,则再测
              LODSB                             ;将字符取到 AL 中
              OUT   56H,AL                      ;输出字符
              LOOP   NEXT-OUT                   ;输出一个字符
```

对上面的程序作几点说明:

① 程序中用 ES、DI 指向输入缓冲区,CX 作为控制循环次数的寄存器,一开始,CX 中设置为最大字符行的长度。

② 方向标志 DF 清 0 是为了使 STOSB 指令和 LODSB 指令在执行时,地址值自动加 1。

③ NEXT_IN 标号后面的 3 条指令是用来测试接口的状态寄存器的,如果状态寄存器的值表明端口未准备好,则用循环测试来等待,直到规定的状态位为 1 才退出循环。

④ 奇偶检验是通过把输入字符和 0 相"或"之后,再对 P 标志进行判断来实现的,这样做很简单,不需要另外再设计子程序进行校验,校验工作完成之后,再用结果和 7F 相与,以清除校验位,再送到输入缓冲区。

⑤ 当用 CMP 指令判断遇到的字符为回车符 0DH 时,程序紧接着再附加一个换行符

0AH，并存入缓冲区，这样做是为了实现程序的设计要求。

利用查询方式进行输入/输出操作时，若系统中对多个外设利用查询方式实现输入/输出，那该怎么处理呢？通常是利用轮流查询的方式来检测接口的状态位。

假设一个系统中有 3 个输入设备，那么，可以用下面的程序来实现轮流查询方式的输入操作。

```
TREE_IN:    MOV   FLAG,0          ;清除标志
INPUT:      IN   AL,STAT1         ;读入第一个设备的状态
            TEST   AL,20H         ;是否准备就绪
            JZ   DEV2             ;否，则转 DEV2
            CALL   PROC1          ;如准备就绪，则调 PROC1
            CMP   FLAG,1          ;如标志被清除，则输入另一个数
            JNZ   INPUT
DEV2:       IN   AL,STAT2         ;读入第二个设备的状态
            TEST   AL,20H         ;是否准备就绪
            JZ   DEV3             ;未准备好，则转 DEV3
            CALL   PROC2          ;如准备好。则调用 PROC2
            CMP   FLAG,1          ;如标志被清除，则输入另一个数
            JNZ   INPU
DEV3:       IN   AL,STAT3         ;读入第三个设备的状态
            TEST   AL,20H         ;是否准备就绪
            JZ   NO-INPUT         ;未准备好，则转 NO-INPUT
            CALL   PROC3          ;如准备好，则调用 PROC3
NO-INPUT:   CMP   FLAG,1          ;如标志被清除，则输入另一个数
            JNZ   INPUT
            ⋮
```

对上面的程序，作如下几点说明：

① 在程序中，对状态寄存器没有赋于具体地址，而用标号 STAT1、STAT2 和 STAT3 来表示。

② PROC1、PROC2 和 PROC3 是 3 个执行输入操作的子程序，这里没有具体列出，

③ 3 个接口的状态寄存器均用第 5 位来作为输入准备好标志位，所以程序中用 20H 去测试状态寄存器的值，再对测试结果作判断，如果结果为 0，则表示未准备好，于是再测下一个接口的状态。

④ 程序中设置了一个标志 FLAG，实际上，这是一个内存单元，将它作为标志单元来用。有了 FLAG 标志后，就可以使第一个输入设备有比较高的优先级，第二个设备次之，第三个输入设备最低。FLAG 开始设置为 0，只有当第一个输入设备结束输入过程时，才使 FLAG 设置为 1。这样，比如第一个输入设备要输入 10 个字符，那么，当输入第 1、2、3、……9 个字符时，FLAG 一直保持为 0，所以，程序一直在 INPUT 标号段运行，当输入第 10 个字符后，FLAG 置 1，于是进入 DEV2。另外一种情况，如果开始第一个输入设备没有准备就绪，那么，转到 DEV2 并且从第二个输入设备输入一个字符后，又会立刻回到 INPUT 程序段。读者从程序中

可以想到，只有在第一个设备未准备好，并且第二个设备也未准备好的情况下，才会转到DEV3 程序段，对第三个输入设备作状态测试，如果状态表明已准备好，则再执行 PROC3 子程序进行输入。

上面的例子可以看到，利用轮流查询方式时，可以通过程序的优先级来决定设备的优先级。根据这样的思想，当系统中有更多的设备时，仍可以安排一个优先级链。

当然，也可以使系统中几个设备处于完全等同的地位，即没有优先级，这种方法叫循环查询法。下面就是实现使 3 个设备处于相同优先级的循环查询程序，在此程序中，FLAG 标志只在循环底部受到检测，如果 FLAG 为 1，则退出循环而不再进行任何输入。可见，在这个程序中，FLAG 作为退出所有 3 个设备输入过程的标志来用了。具体程序如下：

```
INTREE：    MOV  FLAG, 0        ；清除标志
INPUT：     IN  AL, STAT1       ；读入第一个设备的状态
            TEST  AL, 20H       ；测试状态是否准备好
            JZ  DEV2            ；未准备好，则转 DEV2
            CALL  PROC1         ；如准备好，则转 PROC1
DEV2：      IN  AL,STAT2        ；读入第二个设备的状态
            TEST  AL,20H        ；测试是否准备好
            JZ  DEV3            ；如未准备好，则转 DEV3
            CALL  PROC2         ；如准备好，则调转 PROC2
DEV3：      IN  AL,STAT3        ；读入第三个设备的状态
            TEST  AL,20H        ；测试是否准备好
            JZ  NO-INPUT        ；如未准备好，则转 NO-INPUT
            CALL  PROC3         ；如准备好，则调 PROC3
NO-INPUT：  CMP  FLAG,1         ；如标志仍为 0，则继续进行输入测试
            JNZ  INPUT
            ⋮
```

无论是无条件的程序控制 I/O，还是查询式的程序控制 I/O 的接口都比较简单，这是程序控制 I/O 的优点。但是程序控制的输入/输出有两个突出的缺点：浪费 CPU 时间，实时性差。

从所给出的例子可知，CPU 必须等待状态位变为有效状态（外设为传送数据准备好）之后才能进行 I/O 传送，这可能使 CPU 花费很多时间处于不断查询状态的过程之中。例如，对于键盘输入例子，若每秒钟键入 10 个字符，即每个字符的键入时间为 0.1s，而 CPU 完成一次读入操作至多只需 10μs 时间，因而约有

$$(99990 \div 100000) \times 100\% = 99.99\%$$

的时间未被利用，也即在键盘输入过程中，CPU 有 99.99%的时间用于查询状态，显然在这些时间里，CPU 不能进行其它的操作，任何其它的处理都必须等待键盘输入操作完成之后才能进行。CPU 时间利用率很低是程序控制输入/输出的一个主要缺点。

另外，在实时控制系统中，由于一个外设的 I/O 未处理完毕，就不能处理下一个外设的 I/O，故不能达到实时处理的要求，这是查询式方式的致命缺陷。

为了提高 CPU 的效率以及使系统具有实时性能，通常采用中断传送方式。

二、中断控制的 I/O 方式

所谓中断控制的 I/O 方式，也称中断传送方式，即外设的输入数据准备好或接受数据的锁存器空时，便主动向 CPU 发出中断请求（有关中断的详细工作情况，我们将在第八章中讨论），使 CPU 中断现行程序的执行，转去执行为外设服务的输入或输出程序，服务完毕后，CPU 再继续执行原来的程序。

这样一来，由于引入了中断的概念，CPU 和外设（甚至多个外设）就可同时工作，从而大大提高 CPU 的效率和控制程序执行的实时性。

中断传送时的接口电路如框图 7-8 所示。

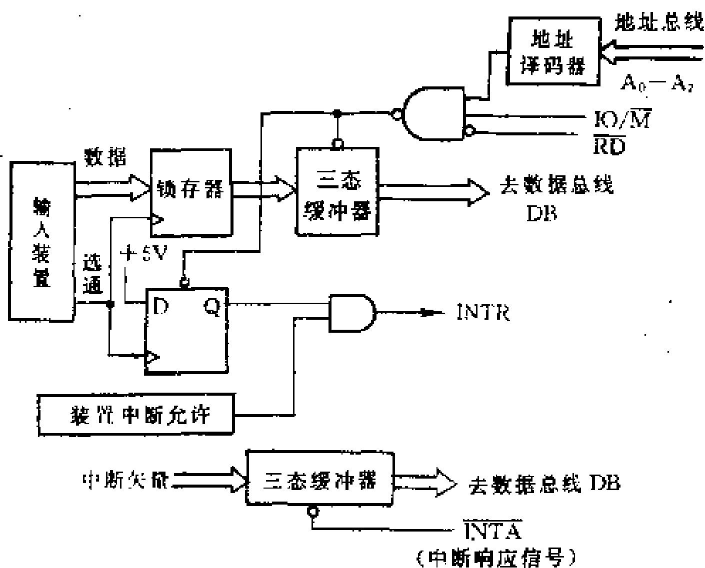


图 7-8　中断传送方式的接口电路

当输入设备输入一个数据时，就发出选通信号，该信号一边把数据存入锁存器，一边又使 D 触发器置“1”，发出中断请求，若中断是开放的，则 CPU 接收了中断请求信号后，就在现行指令执行完后，暂停正在执行的程序，发出中断响应信号 $\overline{INTA}$，由外设将一个中断矢量放到数据总线上，CPU 就转入中断服务程序，同时清除中断请求标志，当中断处理完毕后，CPU 返回被中断的程序继续执行。

程序控制的输入/输出方法是由 CPU 来查询外设的状态，CPU 处于主动地位，外设处于被动地位。而程序中断的 I/O，则是外设处于主动地位，CPU 处于被动地位。只有当外设要传送数据时才向 CPU 发出中断请求信号，实时性比程序控制的 I/O 要好得多。但它仍有其缺点，主要是：

① 为了能接受中断请求信号，CPU 内部要有一些控制线路。

② 利用中断 I/O，每传送一次数据就要中断一次。CPU 响应中断后，每次都要执行“中断处理程序”，而且在其中都要保护现场等，CPU 浪费了很多不必要的时间。故此种传送方式一般较适合于传送少量数据的中慢速外设的场合。对于大量的 I/O 数据可采用高速的直接存贮器存取（DMA）方式。

三、直接存储器存取（DMA）I/O 方式

中断控制的 I/O 虽然克服了查询方式 I/O 的缺点，能够快速地响应 I/O 传送的请求，但是为 I/O 设备的服务仍然是由软件实现，即数据的传送仍由程序完成。为完成一个字节（或字）数据的传送，必须：

(1)暂停主程序，实现程序的转移，即中断响应；

(2)保护和恢复有关寄存器内容；

(3)执行 I/O 操作，并实现内存到累加器再到端口之间的传送；

(4)实现中断返回。

整个过程为传送一个字节或字数据，约需要 110 个时钟，若系统时钟频率为 5MHz，则需要约 22μs 的时间。这对由某些高速的设备或者需要传送成组数据(也称数据块)的设备，已不能满足传送速度上的要求。例如，对于高速的 A/D(模/数)转换器可能要求每隔 1μs 或更短的时间完成一次数据采集，又如对于磁盘装置，数据传送速率是由数据在读写磁头下通过的速度决定的，这个速度通常在每秒 20 万个字节以上，因而要求磁盘与存储器之间传送一个字节的时间应少于 5μs。解决高速传送的方法是采用直接存储器存取，由 DMA 控制器实现在外设与存储器间的直接传送。

所谓直接存储器存取，是在没有 CPU 介入的情况下，由专门的硬件(即 DMA 控制器)直接控制数据的传送。显然，DMA 传送也需要使用系统的数据总线、地址总线和部分控制信号。这时 DMAC 成为系统的主控部件，获得总线控制权，由它产生地址码($A_{19}\sim A_0$) 以及相应的控制信号，而 CPU 不再控制系统总线。用 DMA 传送，传送一个字节(字)数据的时间可减少至 1μs 以下。

依照获得总线控制权的方法不同，DMA 可分为两种类型：暂停 CPU 的 DMA 和不暂停 CPU 的 DMA。

1. 暂停微处理器的 DMA 方式

此类 DMA 操作，CPU 必须暂停任何总线操作，并让出对总线的控制权，直至 DMA 传送结束或完成一个总线操作周期之后，CPU 才能继续控制总线。实现这种方式的方法是 DMAC 向 CPU 发出总线请求信号，CPU 在完成当前的总线周期操作之后，释放对总线的控制(有关的引脚信号处于三态)并发出总线响应信号。当 CPU 工作于最大方式时，实现此类 DMA 操作的一种方法是：由 DMA 控制逻辑产生相应的控制信号，分别送到 8288、8286、8282 以及 8284A，使得 8286 、8282 处于三态，8288 的总线命令信号输出也处于三态，8284A 产生无效(低电平)的$\overline{READY}$信号，使 8086/8088 所执行的当前总线周期处于等待状态。

这类 DMA 又可分为两种情况：周期挪用和数据块传送。如果 DMAC 控制总线的时间为一个总线周期，则称为周期挪用(Cycle-steal DMA)。对于这种操作方式，每次 DMA 操作仅传送一个字节(或字)数据。然后 DMAC 释放对总线的控制。若 DMAC 控制总线的时间超过一个总线周期，用来完成一组数据(即数据块)的传送，在该组数据传送期间，DMAC 一直控制着总线，这种操作方式称为数据块传送(Burst DMA 或 Block transfer)。

2. 不暂停微处理器的 DMA

这类 DMA 操作是利用 CPU 进行内部操作不使用总线时实现的，某些 CPU 能产生一个表示 CPU 是否正在使用总线的信号，当 DMAC 识别出 CPU 不使用总线(即总线处于空闲状态)时便利用总线完成 DMA 操作。由于这种 DMA 是在不暂停 CPU 的情况下进行的，因而每次 DMA 操作只能传送 1 个字节，而不能按数据块传送方式进行 DMA 操作。对于 8086/8088CPU 系统，通常不采用这种 DMA 传送。

下面以周期挪用为例说明 DMA 操作的过程。假定 8086CPU 工作于最小方式，且是将存储器中的数据传送给外部设备，有关部件的连接如图 7-9 所示。在 DMA 操作之前，DMAC 作为系统的一个接口部件，接受 CPU 送来的操作命令，也即对 DMAC 的初始化，以规定传输类型(存储器和外设间)、操作方式(单字节传送)、传送方向、内存的首地址、传送的字节数等。在完成初始化编程之后，DMAC 就准备好进行 DMA 操作，传送一个字节数据的过程如下(在图 7-9 中以圆圈内的序号表示)。

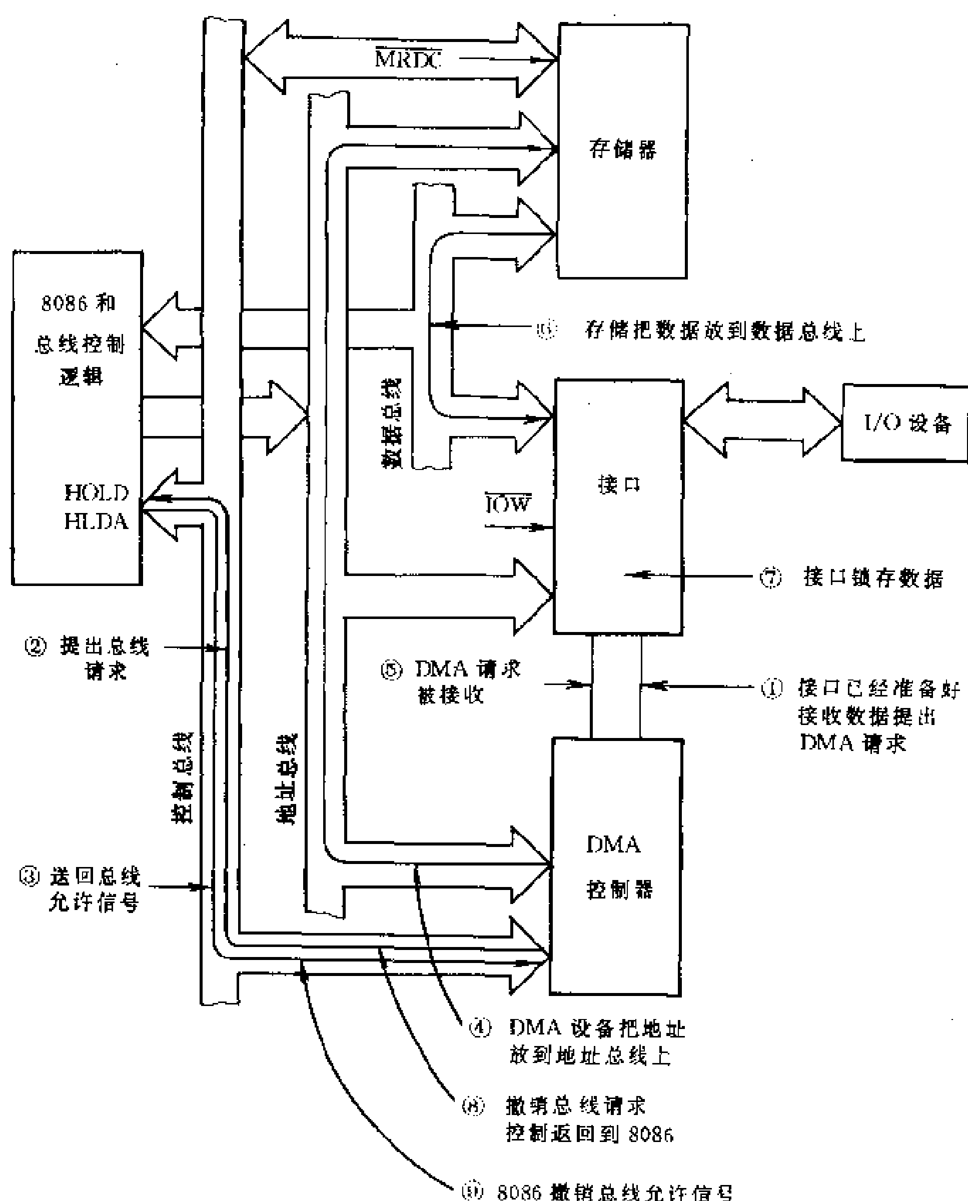


图 7-9 DMA 操作过程示意图

(1) 外设已作好接收数据准备，向 DMAC 发出请求信号 DRQ；

(2) DMAC 向 CPU 发出 DMA 操作请求 HRQ，该请求信号送到 CPU 的 HOLD 信号引脚；

(3) CPU 在完成当前的总线周期操作之后(若总线处于空闲状态，则立即作出响应)，使数据总线、地址总线及部分控制信号处于三态，并发出响应信号 HLDA，指示 DMAC 可以使用总线；

(4) DMAC 将地址($A_{19}\sim A_0$)放入地址总线，该地址用来寻址内存储单元；

(5) DMAC 向 I/O 接口发出响应信号 DACK，该信号可作为接口接受数据的控制条件，有效时，表示允许接口接受数据；

(6) DMAC 向存储器发出读控制信号$\overline{MEMR}$，在该信号的控制下，由 $A_{19}\sim A_0$ 指定的单元的内容送入数据总线；

(7) DMAC 向接口发出写控制信号$\overline{IOW}$，在该信号及 DACK 信号的共同作用下，将数据

总线上的内容锁存到接口中的数据寄存器，并通过接口将数据传送给外设，至此完成了数据传送；

(8) DMAC 撤消对 CPU 的请求信号；

(9) CPU 撤消保持响应信号并恢复对总线的控制。

如果数据未传送完，则又从第二步开始重复所述过程。上述操作相应的时序图如图 7-10 所示。

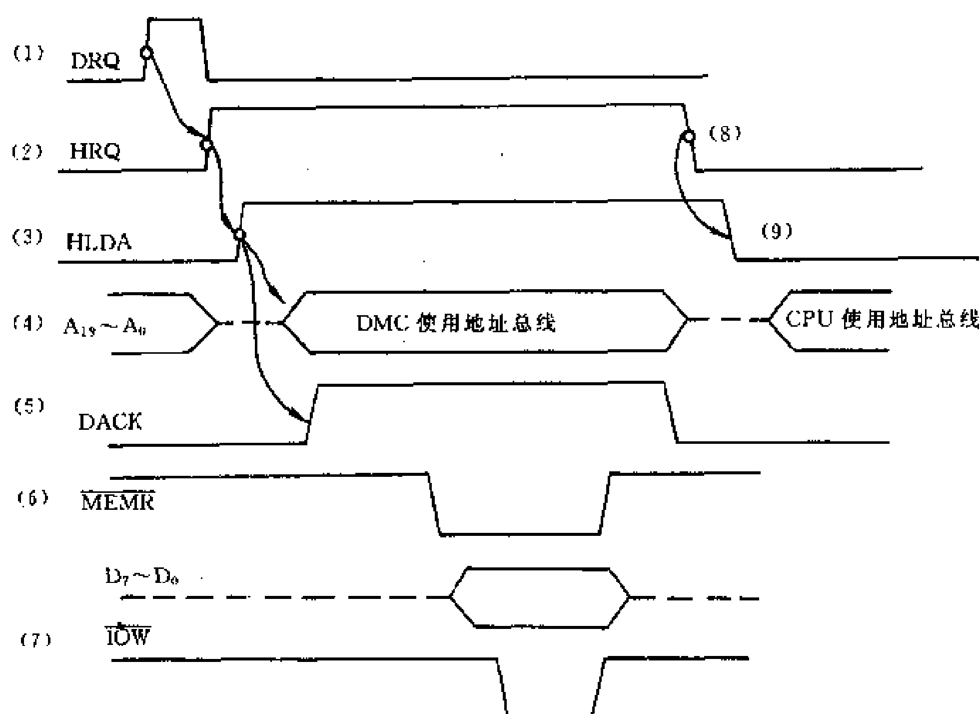


图 7-10 DMA 操作时序

有关 DMAC 芯片的原理、编程和应用将于第四节讨论。

四、IOP 方式

DMA 方式完全由硬件控制数据块传送，DMAC 是可编程的，使用前要由 CPU 对其初始化，结束后要由 CPU 对其进行结束处理，即 DMAC 只能完成 I/O 操作，不能对要传送的数据进行其它处理。为进一步减轻 I/O 对 CPU 的负担，而采用 IOP 方式。INTEL 公司推出了与 8086/8088 CPU 配套使用的高性能通用 I/O 处理器 8089，它可以方便地将 8086/8088 CPU 与 8/16 位的外部设备连结起来相互通信。8089 IOP 具有自己的指令系统，能执行程序，因此它除了能完成输入/输出操作外，还可对传送的数据进行装配、拆卸、变换、校验或比较等多种功能。从而可减轻 CPU 在 I/O 处理过程中的开销，有效地提高整个系统的性能。

系统中配置了 8089IOP 以后，8086/8088 CPU 必须工作在最大方式。当 CPU 需要进行 I/O 操作时，只需在存储器中建立一个规定格式的信息块，设置好需要执行的操作和有关的参数，然后通知 8089 去读这些信息。8089 读取到这些信息后，即可执行 I/O 操作，将数据块以字或字节为单位完成 I/O 的功能。如果在数据 I/O 过程出现差错，8089 还可控制进行重复传送或作必要的处理。在整个数据块的 I/O 过程中，CPU 不必去干予，但可与 8089 并行操作去完成其它功能。

8089IOP 与 8086/8088CPU 协同工作，有两种基本的结构方式：一是本地方式，在这种方式下 8089 与 8086/8088CPU 共享系统总线，可在不增设其它硬件的情况下完成两个 DMA 通道的功能；二是远程方式，这是一种高效率的工作方式，在这种方式下，8089 与 CPU 之间仍然共享系统总线，但 8089 还具有它自己的 I/O 总线，因此 8089 可以不通过共享总线，而只通过自己的 I/O 总线就能访问它自己的 I/O 设备和局部存储器，从而有效地减少 8089 与 CPU 争用总线的现象，以提高 8089IOP 与 CPU 之间并行工作的程度。

8089IOP 中包括两个独立的通道，根据所设定的优先权级别，两个通道可交替工作，完成各自的 I/O 功能。8089IOP 的操作过程是执行通道程序的过程，通道程序由一序列通道指令构成。8089IOP 中设置两个完全独立的通道，各个通道既可执行通道程序，也可以进行 DMA 传送。每个通道都有各自的 I/O 控制器和通道寄存器组。

（一）通道的操作

8089 中的两个通道可并行操作，但是在任何时刻只有一个通道能进行数据传送。因此这是一种宏观的并行，通常称之为并发性，而不是同时性的并行概念。在每一个内部周期结束时，公共控制单元 CCU 都要从两个通道选出一个通道来执行下一个内部周期，这一功能由系统中的优先级机构来完成。

由于优先权级别能反映出各种活动的相对重要性，应该让优先级较高的通道先投入运行。如果当前两个通道的优先级别相同，CCU 将去查询 PSW 中的优先权位 P 的状态，让优先权位高的通道先运行。如果两个通道的优先权位 P 也相等，就让两个通道交替进行操作。

各通道可处于 6 种不同的操作状态，它们的优先权级别如表 7-1 所示。

表 7-1 通道状态的优先级别

通道状态	优先级别	
DMA 传送	1 级	高级
DMA 结束程序	1 级	
通道程序（加链）	1 级	
通道注意程序	2 级	
通道程序（不加链）	3 级	
空闲状态	4 级	低级

从表中可看出，8089 中的通道除了可处于 DMA 传送操作和执行通道程序两种状态之外，还可有其它几种状态。其中 DMA 传送、加链通道程序和 DMA 结束程序具有最高优先级别。DMA 结束程序是当 DMA 传送结束时，通道需要执行的一小段程序，以调整程序块指针 TP 的内容，使通道程序能被恢复执行。通道注意程序是当通道收到注意信号（CA）时，需要执行的一段内部程序，用它来查询通道控制字 CCW 中的内容，并执行 CCW 中的命令，它的优先级是 2 级。这两段程序都是用 8089 指令编写，并事先存放在 ROM 中待用。

通道程序具有 2 种不同的优先级，加链通道程序是 1 级，而不加链程序是 3 级。因此可用加链方式来提高重要通道程序的级别，这可用对通道控制寄存器 CC 中的加链位 C 置位来实现。

在两个通道的优先级和优先权位均相同的情况下，如果某个通道正在进行 DMA 传送，另一通道也要求进行 DMA 传送，可在一个总线周期之间替换另一通道；如果两个通道都要执行通道程序，则必须在一个通道执行完一条通道指令后才能进行替换。由于一条通道指令常常需要几个周期才能完成，因此这种情况的等待时间可能会比较长；此外，如果某个通道正在

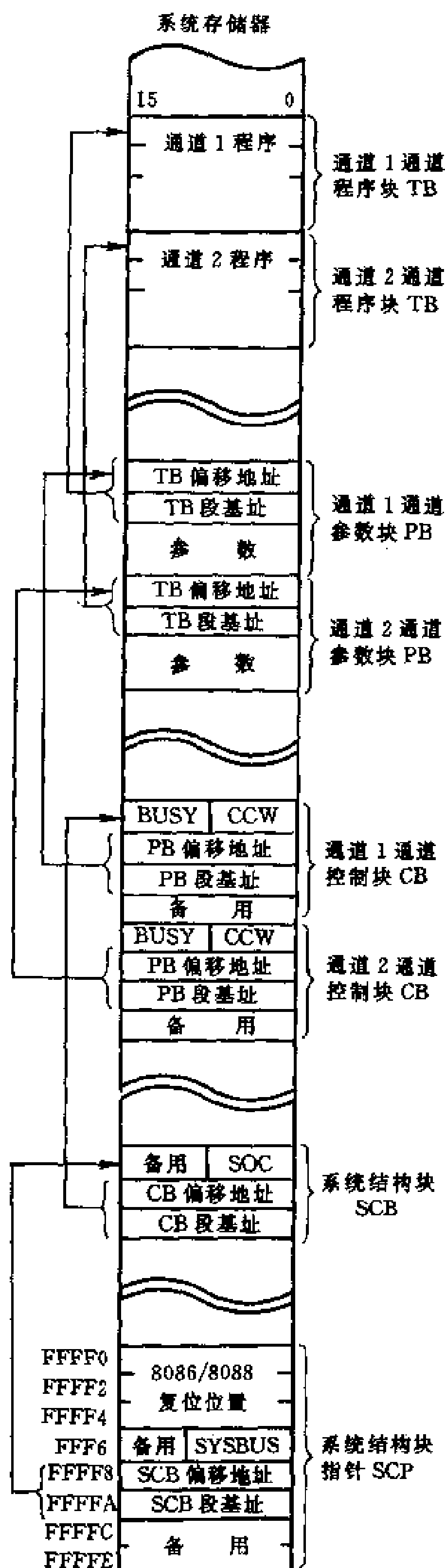


图 7-11　8089 初始化控制块格式

执行 DMA 结束程序或 DMA 注意程序，则不能在一条通道指令执行完被另一通道执行通道程序来替换，只能被另一通道的 DMA 传送来替换。这就是说，DMA 传送具有最高的优先级别。如果两个通道级别相同，都要求进行 DMA，可以以一个总线周期为边界相互替换进行；如果某个通道正处于执行通道注意程序或者是通道结束程序，则另一通道只有 DMA 传送有可能进行替换，尽管加链通道程序的级别比它们高，也无法被替换而优先执行；如果两个通道都要求执行通道程序，那么可以以一条通道指令的执行周期为边界来相互替换。

(二)8089IOP 与 CPU 之间的通信

8089IOP 与 8088CPU 之间的通信是通过共享的系统空间中的存储器进行的，它们之间的通信可分为初始化方式和命令方式两种类型。

1. 初始化方式

初始化方式又称作预置方式，这是 CPU 指派 8089 执行 I/O 操作之前必须进行的通信。初始化过程是从系统加电或复位信号(RESET)开始。实际上是 CPU 在存储器中为 IOP 建立一个信息区，然后通知 IOP 去读取这些信息。这些信息告诉 8089 去完成什么 I/O 操作。

为了对 8089 进行初始化，首先要在共享的系统空间建立一个初始化控制块。它由系统结构块指针 SCP，系统结构块 SCB 和通道控制块 CB 等三部分组成，如图 7-11 所示。其中，系统结构块指针 SCP 包含 16 个字节，固定存放在存储器的高端 FFFF0H～FFFFFH 单元中的地址，再由 SCB 给出通道控制块 CB 在存储器中的地址。

两个通道的通道控制块包含 8 个字节，在存储器中必须连续存放。CPU 与 IOP 之间的通信信息实际上集中在通道控制块 CB 中。其中，忙位(BUSY)占一个字节，它表示该通道是正在进行操作(BUSY 为全“1”字节)，还是等待 CPU 发命令(BUSY 为全“0”字节)。

CPU 通过通道命令字 CCW 命令该通道完成什么操作，如果 CPU 命令该通道执行通道程序，则应由参数块指针给出该通道的参数块 PB

在存储器中的地址。通道参数块的大小取决于所需参数的多少，其中前面 4 个字节给出该通道的通道程序块 TB 在存储器中的地址。通道程序块 TB 中所存放的是要求该通道执行的通道程序，它是用 8089 指令编写的。通道执行通道程序所需要的参数，取自各自的参数块 PB 中。

对于任何一个通道程序来说，可以设置许多不同用途的参数块和程序块，但是任时候只允许有一个参数块和一个程序块与某通道相连。8086/8088CPU 对 8089IOP 的初始化过程实际上是从 CPU 向 IOP 发出通道注意命令 CA 开始的。在 8089IOP 中某个通道被初始化之前，CPU 应准备好初始化控制块，并将“FF”装入所选通道的通道控制块 CB 内的 BUSY 字节中，而另一个通道的 BUSY 字节应置成“00”。IOP 在收到 RESET 信号后停止一切通道操作，等待 CPU 发出的通道注意信号。

在远程方式下，可将 IOP 定义为主设备，这时要求选择通道 1，即 SEL＝0。若将 IOP 定义为从设备，则应选择通道 2，即 SEL＝1。

从 CPU 向 IOP 发出命令 CA 开始，CPU 就与 IOP 并行操作，CPU 就开始测试通道 1 的 BUSY 字节是否为“0”，IOP 读出初始化控制块中的有关信息，进入初始化过程。直到通道 1 初始化过程结束，就将通道 1 的 BUSY 单元清“0”。当 CPU 查询到通道 1 的初始化过程结束，便可进入对其它通道进行初始化的过程。

如果 IOP 被初始化为主设备，它将能立即获得总线的使用权；如果为从设备，就将通过 $RQ/\overline{GT}$信号线向 CPU（本地方式）或向其它处理器（远程方式）发出使用总线的要求。一旦 IOP 得到总线的使用权，它就假定系统总线为 8 位，到系统结构块指针 SCP 中的 FFFF6 单元取出 SYSBUS 字节，该字节可告诉 IOP 系统总线的实际宽度，由 SCB 地址指针可找到系统结构块，从 SCB 中可获得系统操作命令 SOC 字节。由它告 IOP，I/O 总线的宽度和请求/授予的方式。从 SCB 中得到通道控制块 CB 的物理地址后，即可将该通道控制块中的 BUSY 字节清“0”，初始化过程到此结束。

2. 命令方式

8089IOP 初始化过程结束后，重新进入等待通道注意命令 CA 的状态，从此以后的 CA 命令是向通道 1(SEL＝0)或通道 2(SEL＝1)发出的操作命令。当任一通道经初始化以后又收到了 CA 命令，就立即将其 CB 中的 BUSY 字节置为全“1”，并从通道控制块 CB 发出通道命令字 CCW，其具体格式如图 7-12 所示，CCW 中的命令字段 CF 占 3 位，可定义 6 种通道命令，其操作示意图如图 7-13 所示。

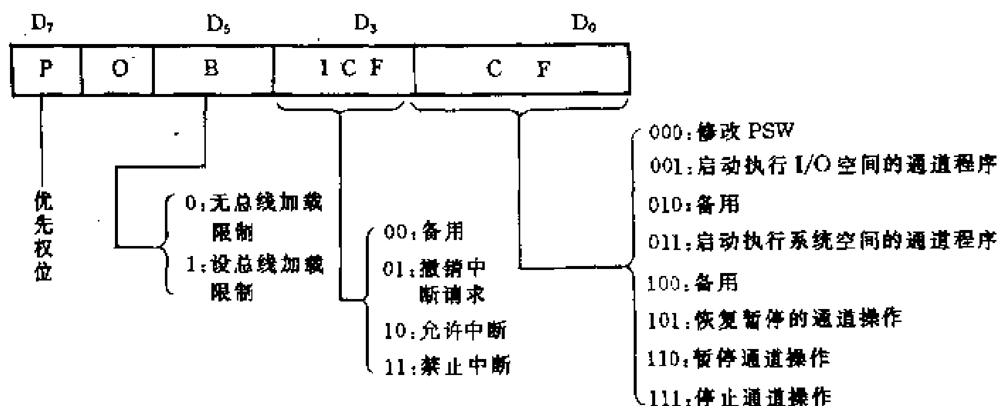
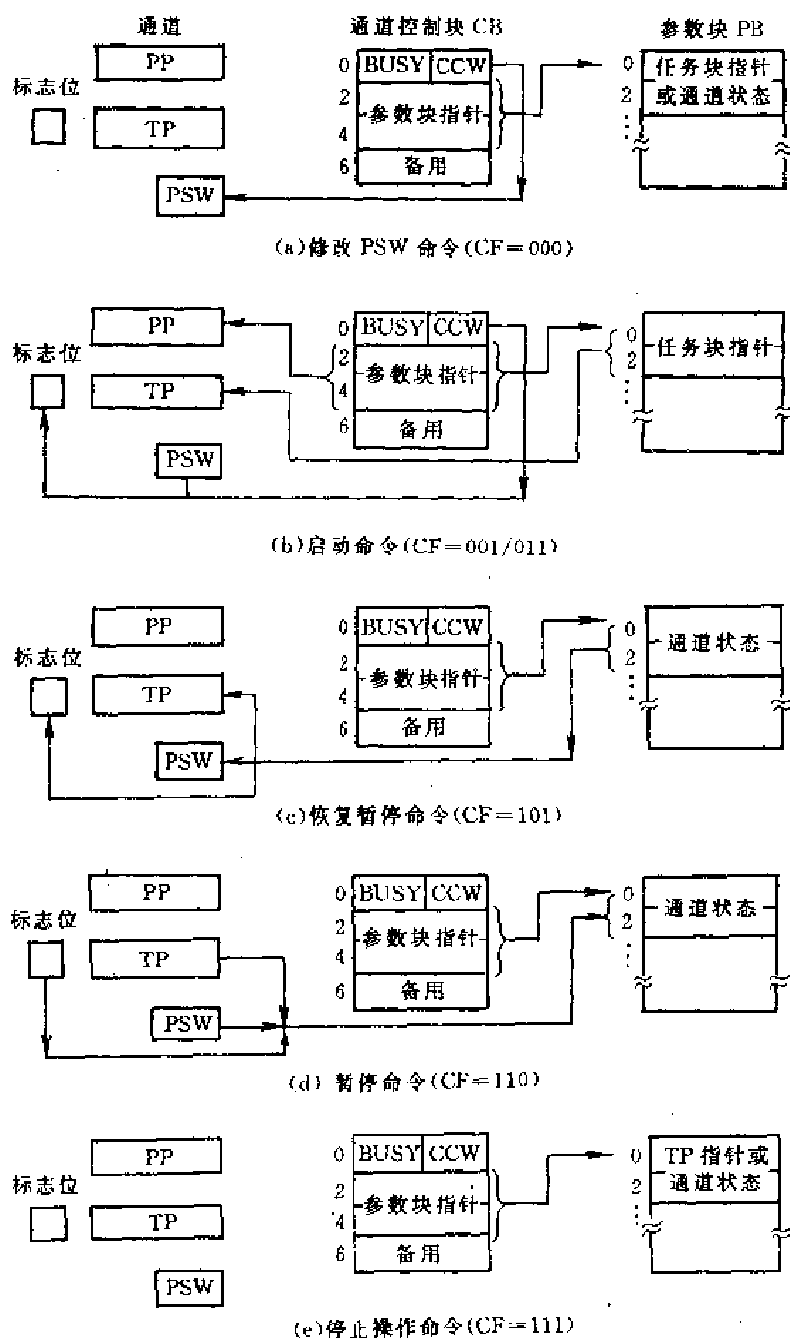


图 7-12 CCW 格式



(a)修改 PSW 命令(CF=000)

(b)启动命令(CF=001/011)

(c)恢复暂停命令(CF=101)

(d) 暂停命令(CF=110)

(e)停止操作命令(CF=111)

图 7-13 通道命令操作示意图

(1) CF=000,则用 CCW 中的 B 位和 P 位修改 PSW 中的总线负载,限制位 B 和优先权位 P;用 ICF 给定的状态修改 PSW 中的中断控制位 IC。若 ICF=10,把 IC 修改为"0",允许中断;若 ICF=11,把 IC 修改为"1",禁止中断;若 ICF=01,把 IC 清"0",表示 CPU 已经响应过该中断,可撤消该中断请求。

(2)CF=001/011,为启动通道执行通道程序的命令。CF=001 时,将程序块指针 TP 的标志位置"1",表示通道程序存放在 I/O 空间;CF=011 时,TP 标志位置"0",表示通道程序存放在系统空间。通道接收到启动命令后,应执行如下操作。

① 根据通道控制块 CB 提供的通道参数块 PB 指针,形成 PB 首地址的 20 位物理地址,并将它装入参数块指针寄存器中。

② 若 CF=001,即通道程序在 I/O 空间,将 PB 中的程序块的偏移地址直接装入程序块指针 TP 的低 16 位中(高 4 位无效),即为通道程序的入口地址。

③ 若 CF＝011，则表示通道程序在系统空间，应根据 PB 中提供的程序块指针形成通道程序入口的 20 位物理地址，并装入 TP 寄存器中。

④ 根据 CCW 中的 ICF 字段，B 字段和 P 字段修改程序状态字 PSW 内容，修改原则与 CF＝000 时相同。

⑤ 由 TP 提供地址，逐条执行通道程序。

(3)CF＝110，为暂停通道操作命令。通道应将当前的通道状态（包括 PSW、程序块指针 TP 及其标志位）保存到通道参数块 PB 的前 4 个字节中。在通道程序未执行前，这 4 个字节用来存放指向程序块 TB 的指针，一旦通道程序开始执行，它就不需要继续保留，完全可用来暂存当前的通道状态，被称作通道状态保护区，其存放格式如图 7-14 所示。

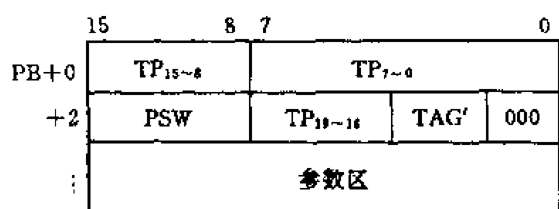


图 7-14　通道状态保护区格式

(4)CF＝101，为恢复暂停通道操作命令。通道执行该命令只需将通道状态保护区暂存的信息送回 PSW、TP 及其标志位中，即可从断点继续执行被暂时“挂起”的通道程序。

(5)CF＝111，为停止通道操作命令。通道执行该命令，把它的 BUSY 字节清“0”，通道操作立即停止，进入空闲状态。

(三) 8089IOP 的 DMA 传送

8089 IOP 不仅能以执行通道程序的方式来完成 I/O 操作，而且还能像 DMA 控制器一样以 DMA 方式高速传送数据。DMA 传送的源操作数和目标操作数可来自系统空间，也可来自 I/O 空间，因此可以在存储器与存储器之间，存储器与 I/O 端口之间，I/O 端口与 I/O 端口之间进行数据块的传送。当采用给定的字节数来结束 DMA 操作时，则所能传送的数据块最大为 64K 字节；若采用其它结束方式，对所传送的数据块大小没有任何限制。

如果原来 8089IOP 在执行通道程序，只需向 8089 发出一条“进入 DMA 方式”的指令（XFER），8089 将在当前一条通道指令执行完后，立即停止执行通道程序而进入 DMA 传送工作方式。每一个 DMA 传送周期至少包含 2 个总线周期，第一个总线周期读取一个源操作数字节或字到 IOP，第二个总线周期将数据从 IOP 传送到目标操作数中。在传送数据过程中，还可对数据进行加工，例如翻译、屏蔽和比较等操作。

一旦 DMA 传送结束，IOP 将自动恢复执行通道程序，并可利用通道程序检查 DMA 传送结果。如发现错误，通道程序能自动控制将数据块重新传送一遍。

启动通道进行 DMA 传送之前，通道程序必须对外设接口部件进行初始化，通道本身也要作好必要的准备。

1. 外设接口的初始化

DMA 传送和一般 I/O 操作一样，所涉及到的外设都是通过可编程的接口相连，进行 DMA 传送前，必须对它们进行初始化编程。在 8089IOP 中，可用通道程序来对它们初始化，而不必 CPU 干预。通常是利用一条通道指令将通用寄存器 GC 中的有关命令和参数装入相应接口芯片的命令口/状态口，这条指令通常安排在“进入 DMA 方式”指令（XFER 指令）之后，待相应接口芯片接收到这一命令，就立即开始 DMA 传送过程。

2. 通道的准备工作

启动通道执行 DMA 传送之前，应利用通道程序将所需的控制信息装入相应的通道寄存

器，并设置总线的逻辑宽度。

必须传送的控制信息有源指针、目标指针、通道控制命令和总线逻辑宽度等，有些情况还需要传送字节数、翻译表指针和屏蔽/比较值等，这些控制信息应装入的寄存器如表 7-2 所示。

表 7-2 寄存器约定功能

	控制信息	通道寄存器		控制信息	通道寄存器
必	源指针	GA 或 GB	可	传送字节数	BC
	目标指针	GB 或 GA		翻译表指针	GC
	通道控制	CC		屏蔽/比较值	MC
需	总线逻辑宽度	RESET 后用 WID 指令完成	选	总线逻辑宽度	非 RESET 后用 WID 指令完成

3. DMA 传送

一次 DMA 传送过程大体可分为三个阶段。第一阶段是 DMA 启动阶段，第二个阶段是连续执行 DMA 周期完成数据块传送阶段，第三阶段是 DMA 结束阶段。

(1) DMA 启动阶段

进入 DMA 启动阶段的标志是在通道程序中安排一条 XFER(进入 DMA 方式)指令，为了给通道一些准备时间，需要在 XFER 指令后再安排一条指令。IOP 在执行完该指令后，才真正停止执行通道程序，而进入 DMA 传送阶段。至于这一条指令应该安排什么指令可随具体情况而定。

① 如果这次 DMA 传送要求完成 I/O 端口与 I/O 端口之间或 I/O 端口与存储器之间的数据传送，那么可根据所涉及到的 I/O 端口的要求来设定。

若所涉及的 I/O 端口需要进行初始化后才能开始 DMA 操作，那么就可把最后一条初始化指令安排在 XFER 指令的后面，待 IOP 执行完这最后一条初始化指令后，立即进入 DMA 传送阶段。

② 如果这次 DMA 传送要求在存储器与存储器之间进行数据传送，那么从原理上说，在 XFER 指令后安排任意一条 8089 指令都可以，但是由于 DMA 传送过程中需要使用一些通用寄存器作为指针，并经常要对它们进行修改，因此一些能修改通用寄存器 GA、GB 和 GC 等内容的指令不能安排，还有能将 BUSY 标志清“0”的 HLT 指令也不能安排。除此之外，没有其它的限制。

(2) DMA 传送阶段

DMA 传送过程是连续执行 DMA 周期的过程，每一个 DMA 周期可以传送一个字节或一个字的数据，因此 DMA 周期是完成 DMA 传送的基本周期。在 DMA 传送期间，根据所设置的源总线和目标总线的逻辑宽度，可对传送的数据进行装配/拆卸，以保证能有效地进行 DMA 操作。

依据四种不同的总线逻辑宽度情况，DMA 周期内的数据传送过程如表 7-3 所示。

表 7-3 DMA 周期内的数据传送

数据地址（源→目标）	8 位→8 位	8 位→16 位	16 位→8 位	16 位→16 位
偶地址→偶地址	B→B	B/B→W	W→B/B	W→W
偶地址→奇地址	B→B	B→B	W→B/B	W→B/B
奇地址→偶地址	B→B	B/B→W	B→B	B/B→W
奇地址→奇地址	B→B	B→B	B→B	B→B

在 DMA 传送阶段进行数据传送时，还要考虑到可采用三种不同的同步方式，它们是非同步方式、源同步方式和目标同步方式。由于 DMA 请求信号(DRQ)通常是由 I/O 接口发出的。因此当进行从 I/O 端口至存储器传送数据时，应采用源同步方式；当进行从存储器到 I/O 端口数据传送时，应采用目标同步方式；当进行从存储器到存储器数据传送时，应采用非同步方式。

当 8089 接收到 DRQ 信号后，需要等待一段时间，才给出响应信号 DACK，通常把这一段等待时间称作 DACK 延迟时间。如果另一个通道是空闲的，则 DACK 延迟时间一般不超过五个时钟周期。而如果两个通道同时有效，则最大 DACK 延迟时间可达 12 个时钟周期，待 DACK 有效后，DMA 周期才真正开始。DMA 周期内的操作流程图如图 7-15 所示。

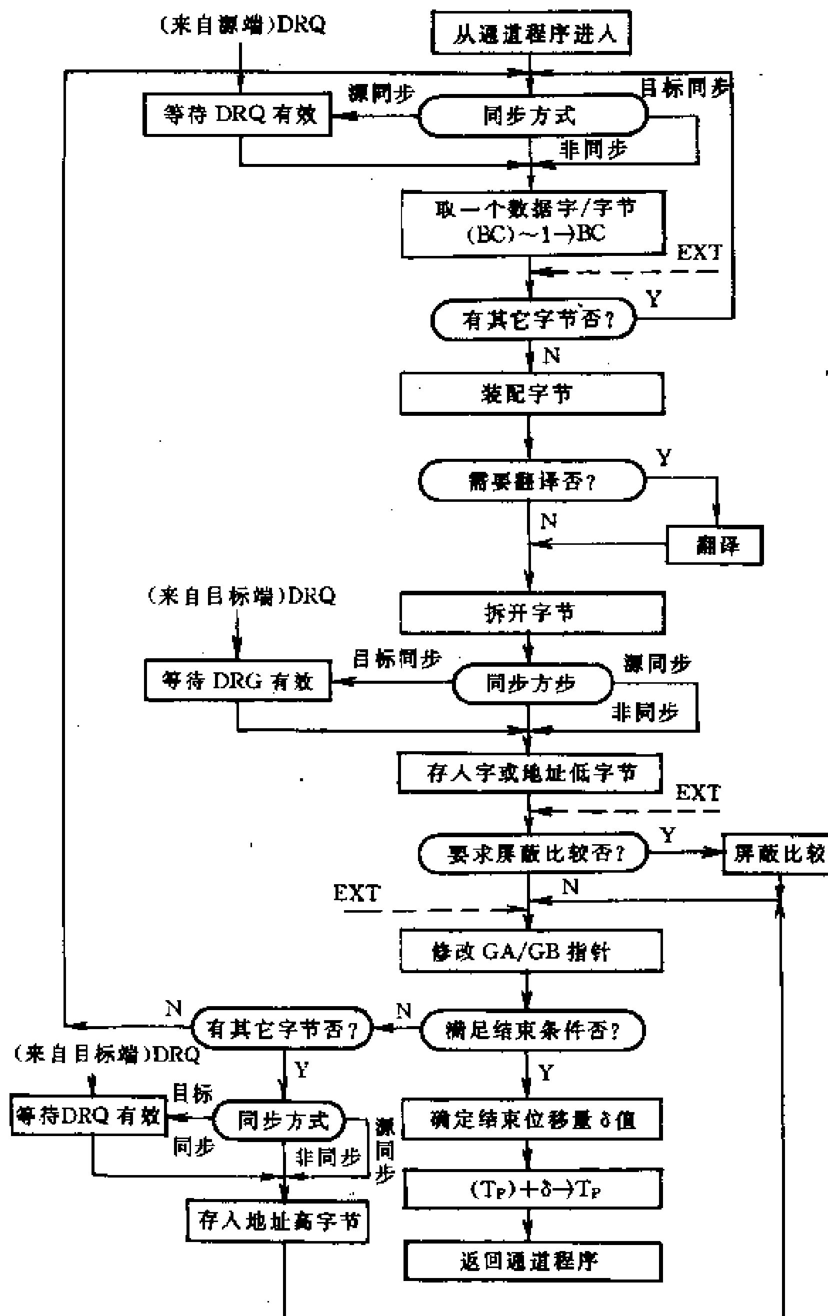


图 7-15 DMA 周期操作流程图

当要求停止正在执行的通道程序进入 DMA 传送方式时，首先要查询所采用的同步方式。如果采用源同步方式，那么必须等待源端发来 DRQ 请求信号。

如果采用非同步方式，则可立即进入 DMA 的取总线周期，通过 8/16 位总线取出一个数据字节/字到 IOP 中。如果是进行 B/B→W 传送，则要连续执行两个取总线周期，取出两个字节，并装配成一个数据字。如果是进行 W→B/B 传送，则要将取得的数据字拆开成两个数据字节。若还要求对传送的数据进行翻译，则应按字节完成翻译功能。

如果采用目标同步方式，则必须等待目标端发来 DRQ 请求信号后，才能向目标端传送数据。若通过 CC 寄存器的 TMC 字段设置了屏蔽比较结束条件，则要对传送的数据进行屏蔽比较操作，并建立比较结果，修改 GA/GB 寄存器内容，使之指向下一数据地址，然后检查是否满足结束条件。只有不满足结束条件，才可进入下一个 DMA 周期，继续完成后继数据的传送。上述过程一直进行到满足结束条件，则立即将 CC 寄存器给定的结束位移量值。与 TP 寄存器内容相加，控制返回到通道程序的断点继续执行。

(3) DMA 结束阶段

由 CC 寄存器给定，DMA 传送可有三种不同的控制结束方式，它们是外部信号结束方式，字节计数结束方式和屏蔽比较结束方式。

三种结束方式可在不同的 DMA 传送中单独使用，也可在一次 DMA 传送中同时使用，只要满足任一结束条件，都可结束 DMA 传送操作。对于外部信号结束方式，则要求在每一个总线周期结束时都要查询 EXT 信号是否有效。若在取总线周期查询到 EXT 有效，则立即结束 DMA 传送，而不再进入存总线周期。而如果在第一个取/存总线周期查询到 EXT 有效，则不再进入第二个取/存总线周期。

如果忘记设置结束条件或所设置条件永远不能满足，DMA 传送将一直进行下去，这在使用中要特别注意。

第四节　DMA 控制器 Intel 8237

为了能够实现高速率传送数据，人们提出了直接存储器存取(DMA)这种方法。本节将以广泛使用的 DMA 控制器 Intel 8237 为例分析 DMA 的一般原理及工作过程，最后举例说明其使用方法。

8237 是高性能的可编程 DMA 控制器，工作在 5MHz 时钟下的 8237A－5 传输速率可达 1.6M字节/秒。每片 8237 内部有四个独立的通道，每个通道寻址及字节计数都可达 64K。它们可以分时地为四个外部设备实现 DMA 操作，可以用来实现内存到端口，端口到内存及内存到内存之间的高速数据传送。在 PC 机中，利用 8237 的传送速率仅仅是 476K 字节/秒。

一、8237 的结构和引脚

1. 8237 的结构

8237 每一通道内包含四个 16 位的寄存器，即基地址寄存器、基字节数寄存器、当前地址寄存器和当前字节计数器。它们决定了 DMA 访问的存储器地址及传输数据的字节数。每个通道内还有一个 6 位的模式寄存器，用来在初始化编程时选定通道的工作方式。

每片 8237 中，四个通道还共用一个 8 位的命令寄存器、一个 8 位的状态寄存器、一个 4 位的屏蔽寄存器和一个 4 位的请求寄存器等。整个 8237 的内部寄存器如表 7-4 所示。

表 7-4　8237 内部寄存器

寄存器名称	位长	数量
基地址寄存器	16	4
基字节数计数器	16	4
当前地址寄存器	16	4
当前字节数计数器	16	4
临时地址寄存器	16	1
临时字节数计数器	16	1
状态寄存器	8	1
命令寄存器	8	1
临时寄存器	8	1
模式寄存器	6	4
屏蔽寄存器	4	1
请求寄存器	4	1

2. 8237 的引脚

8237 的引脚信号见图 7-16 所示。

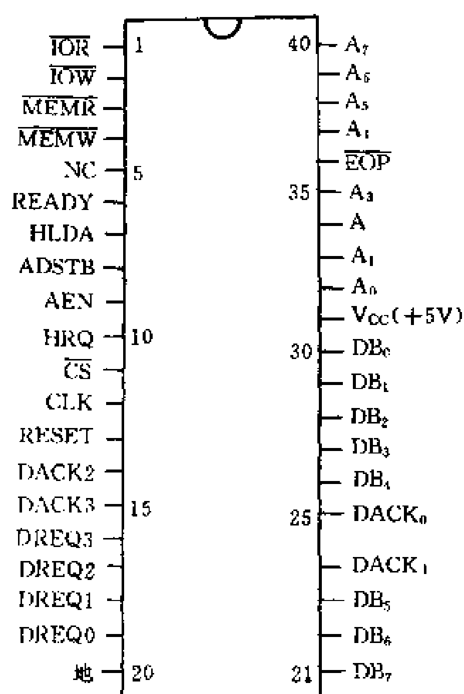


图 7-16　8237 引线图

CLK　输入，时钟信号。用来控制 8237 内部操作的时序及数据传输的速率，对于 8237A-5，可用 5MHz。

$\overline{CS}$　输入，片选信号，低电平有效。当 CPU 控制总线时，用这个信号来选中 8237 进行 I/O 读/写操作。

RESET　输入，复位信号，高电平有效。复位后，除屏蔽寄存器被置 1(四个通道全被屏蔽)外，其余所有的寄存器都被清零。

REDAY　输入，准备好信号，高电平有效。表示进入 DMA 的外设已为读写准备好，否则在总线周期中要插入等待状态 S_W。

AEN　输出，DMA 地址允许信号，高电平有效。当 AEN 呈高电平时，允许 DMA 控制器送出地址信号而禁止 CPU 地址线接通系统总线；只有当 AEN 为低电平时，才允许 CPU 控制系统总线上的地址信号。

ADSTB　输出，地址选通，高电平有效。8283 的数据线 DB_7～DB_0 供 DMA 地信号 A_{15}～A_8 分时使用，当 ADSTB 信号有效时，DB_7～DB_0 上出现的是 DMA 地址高字节，被此信号选通进入外部锁存器(例如 LS373)。

MEMR　输出，DMA 存储器读信号，低电平有效。读出的数据可以直接传送给外设。

MEMW　输出，DMA 存储器写信号，低电平有效。写入的数据可以直接来自外设。

$\overline{IOR}$ 双向，I/O 读信号，低电平有效。当 8237 作为从属器件时，$\overline{IOR}$信号作为输入，配合片选信号$\overline{CS}$，由 CPU 读 8237 内部寄存器。当 8283 作为主控器件时，输出$\overline{IOR}$信号，以读取外设的数据而写入存储器。

$\overline{IOW}$ 双向，I/O 写信号，低电平有效。和$\overline{IOR}$一样其信号传输方向视 8237 在总线上的地位而定。

$\overline{EOP}$ 双向，DMA 过程结束信号，低电平有效。若 8237 中任一通道进入 DMA 过程，当其字节计数结束时，即输出$\overline{EOP}$有效。若 DMA 计数未完，但外部输入一个有效的$\overline{EOP}$信号，则强制结束 DMA 过程。只要$\overline{EOP}$信号有效就会复位内部寄存器。在 PC/XT 主板上，$\overline{EOP}$输出经过反相后产生系统总线上的 T/C 信号。

$DREQ_3 \sim DREQ_0$ 输入，外设对 8237 四个通道分别提出的 DMA 请求信号，其有效极性可以编程设定。在固定优先权情况时，$DREQ_0$ 优先级最高，然后依次下降，$DREQ_3$ 最低。当多个通道同时申请时，8237 只能选择优先级最高的一个通道响应。DREQ 信号须保持到响应信号 DACK 有效以后才可撤消。在复位之后，DREQ 是高电平有效。在 PC/XT 机中 DREQ 也被编程为高电平有效。

$DACK_3 \sim DACK_0$ 输出，8237 给外部的响应信号。其有效极性也可以编程设定，复位后规定低电平有效，在 PC/XT 机中也是低电平有效。

HRQ 输出，8237 对 CPU 的总线请求信号，高电平有效。8237 接受了任何一个通道有效的 DREQ 请求之后，就会产生 HRQ 信号。

HLDA 输入，CPU 回答 8237 的总线响应信号，高电平有效。

上面四种信号，以 8237 为中介，分别向微型计算机和向外设形成了两套握手信号 。在一次 DMA 工作过程中，首先是外部向 8237 某一通道提出 DREQ，如果有效，则 8237 向 CPU 提出 HRQ。等 CPU 给 8237 发回响应 HLDA 后，8237 就给外设发出响应 DACK，至此 DMA 传送开始。

$DB_7 \sim DB_0$ 双向数据线。CPU 用其对 8237 内部寄存器进行读写。在 DMA 传送开始时，存储器地址的 $A_{15} \sim A_8$ 经过 $DB_7 \sim DB_0$ 线送出锁存。在同时利用通道 0 和通道 1 进行存储器到存储器的传送时，从源存储单元读出的数据还要经过数据线进入 8237 内部暂存，然后再经数据线写入目的存储单元。

$A_3 \sim A_0$ 双向地址线。CPU 输出的 $A_3 \sim A_0$ 用来选择访问 8237 内部的寄存器。8237 输出的 $A_3 \sim A_0$ 是被读写的存储单元地址的最低 4 位。

$A_7 \sim A_4$ 三态输出，在 DMA 时用来输出被读写存储单元地址的 $A_7 \sim A_4$。

二、8237 的工作时序

8237 的工作时序如图 7-17 所示。

为了区别于 CPU 的时钟周期，DMA 的每一个时钟周期称为一个 S 状态。

1. S_1 状态是空闲状态。在进入 DMA 传输之前，8237 一直处在连续的 S_1 状态。这时 8237 作为从属器件，可以接受 CPU 的编程写入或读出。在 S_1 状态中 8237 要不断地采样各 DREQ 信号。若有 DREQ 信号（一个或多个）产生，且经过屏蔽逻辑及优先级排队后仍有效，在 S_1 的上升沿产生 HRQ 信号，向 CPU 发出总线请求，同时结束 S_1 状态，进入 S_0 状态。

2. S_0 状态中 8237 等待 CPU 的总线响应信号 HLDA，在 HLDA 信号有效之前，8237 一直重复 S_0 状态。S_0 状态中的 8237 还是从属器件，可以接收 CPU 的读写。在 S_0 的上升沿检测到

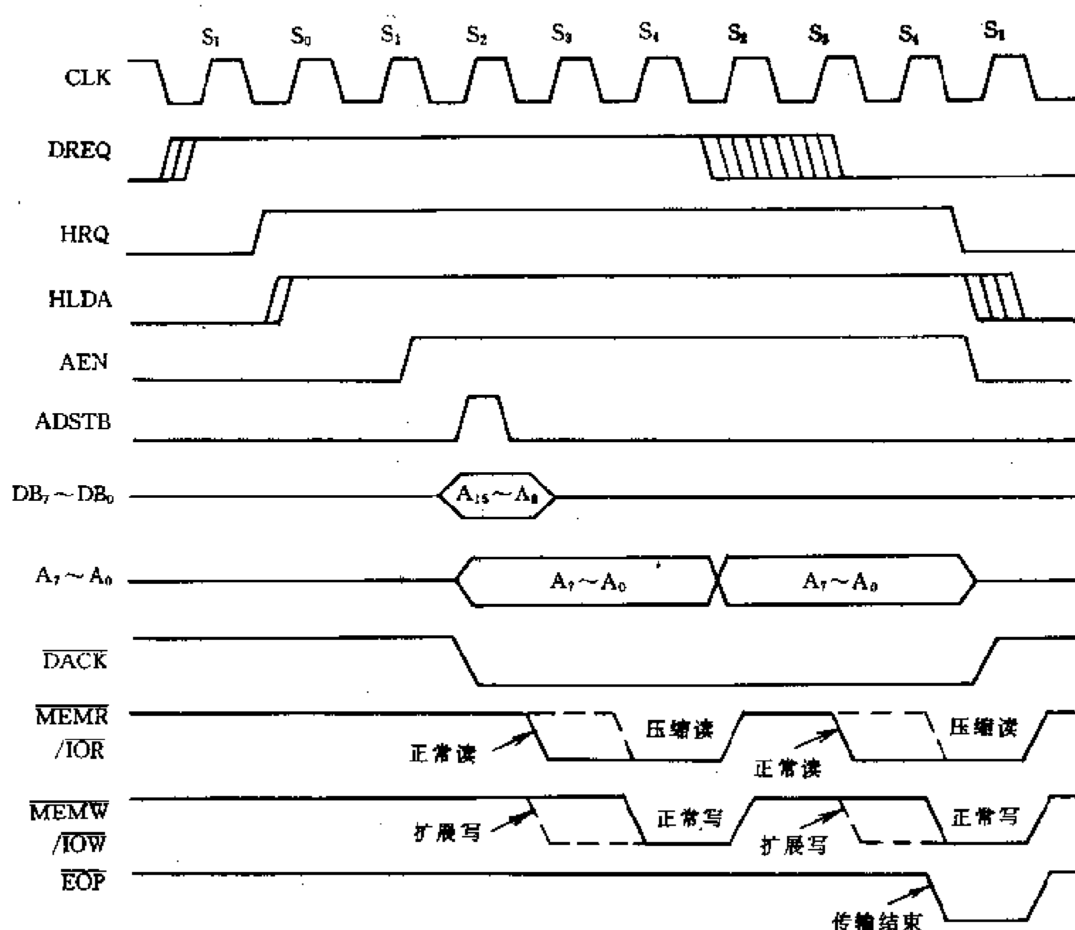


图 7-17　8237 的时序

HLDA 信号有效，则下一状态进入 S_1。真正的 DMA 传送是从 S_1 状态开始。

3. S_1 状态首先产生 AEN 信号，使 CPU 等其它总线器件的地址和系统总线的地址线断开，而使 8237 的地址线 A_{15}～A_0 接通。AEN 信号一旦产生后在整个 DMA 过程中一直有效。S_1 状态还有一个作用是产生 DMA 地址选通信号 ADSTB，将 DB_7～DB_0 上送出的地址信号 A_{15}～A_8 用 ADSTB 的下降沿锁存到外部锁存器中。考虑到在块传送方式中，相邻 字节的高位地址往往是相同的。最极端的情况下，连续传送 256 字节，地址 A_{15}～A_8 才变化一次。因此不必每次都用 ADSTB 信号将一个不变的 A_{15}～A_8 锁存一次。这种情况下，连续下去的 DMA 时序中省去 S_1 状态，不产生 ADSTB 信号，直接从 S_2 状态开始，如图 7-17 中后一部分所示。

4. S_2 状态中 8237 产生 DMA 响应信号$\overline{DACK}$给外设。得到响应的外设，可用$\overline{DACK}$信号代替 CPU 控制总线时的片选信号，使自己在整个 DMA 期间都处于选中状态，同时地址总线上出现所要访问的存储器地址 A_{15}～A_0。

5. S_3 状态产生$\overline{MEMR}$或 IOR 读信号，于是数据线 DB_7～DB_0 上出现被传送的字节。正常的读信号从 S_3 状态一直持续到 S_4 状态，以便使 DB_7～DB_0 上的数据稳定至 S_4 状态写入目的处。另有一种压缩读方式取消 S_3 状态，读信号和写信号同时在 S_4 状态时产生，适用于高速电路。相反若 DMA 传送数据的源或目的电路速度较慢，不能在 S_4 状态前使读出数据稳定，则可以将 8237 的准备好 READY 端变低，使 S_3 和 S_4 之间插入等待状态 S_W 直到准备好 READY 变高才结束 S_W 进入 S_4 状态。在 PC/XT 机中除通道 0 以外都固定地插入一个 S_W。

6. S_4 状态产生$\overline{MEMW}$或$\overline{IOW}$写信号，将 DB_7～DB_0 上的数据写入目的电路，写信号也可以提前到 S_3 状态时产生，这就是所谓的扩展写(超前写)。若是块传输，则在 S_4 结束后又进入 S_1(或 S_2)，继续传输下一字节。若是单字节传输或是块传输的最后一个字节传输完成，则产生传输结束信号$\overline{EOP}$，并撤消总线请求信号 HRQ 释放总线。外部输入的$\overline{EOP}$信号强制 8237 在完成传输当前字节的 S_4 后结束 DMA 过程。

综上所述，在进入 DMA 传输过程之后，传送一个字节一般需要 4 个 S 状态。各个状态的主要作用是：S_1 产生 AEN 信号并锁存存储器地址 A_{15}～A_8。S_2 产生$\overline{DACK}$信号并送出 16 位存储器地址 A_{15}～A_0。S_3 读数据，S_4 写数据。在存储器地址 A_{15}～A_8 不变的情况下，还可以在 S_3 和 S_4 之间插入 S_W。

三、8237 的编程

8237 是高性能的可编程器件。为了讲述方便，对某些主要的编程功能先作一些解释。

8237 的每个通道都有自己的模式寄存器。通过对模式寄存器写入不同的内容，各通道可以独立地选择不同的工作模式和操作类型。

1. 工作模式

在 DMA 传输时，每个通道有四种工作模式供选择。

(1) 单字节传输模式

每次 DMA 过程仅传送一个字节数据，当前字节数计数器减 1，当前地址寄存器作加 1 或减 1 修正，随后撤消 HRQ 信号，退出 DMA，向 CPU 交还总线控制权。即使 DREQ 持续有效，CPU 也要继续执行指令，一个总线周期后才再次响应 DMA 请求，传输下一个字节。这种 8237 和 CPU 轮流控制总线的过程一直进行到要求的字节数全部传输完毕，发出$\overline{EOP}$信号，才最后退出 DMA。由于 DMA 过程不能被另一个哪怕是优先级更高的 DMA 请求所打断(即不会发生 DMA 嵌套)，因此单字节传输模式的一个明显优点是系统总线不至于长时间陷入对某一个 DMA 通道的服务。当某一较低级通道要求传送的字节数较多时，CPU 可频繁地收回总线，有机会对各 DREQ 信号重新作优先级排队，及时响应更高级的 DMA 请求。

(2) 块传输模式

取这种模式的 DMA 通道一旦开始传输，就一个字节接着一个字节进行下去，直至完成预定数据块的字节数才交出系统总线。若需要提前结束其传输过程，可由外面输入一个有效的$\overline{EOP}$信号来强制 8237 退出。显然这种模式的传输效率较高。

(3) 请求传输模式

这种模式也用于成块数据的传输，和上面的块传输模式相比，仅增加了一个功能，即用 DREQ 信号无效来打断传输过程。一般情况下，请求信号 DREQ 保持到响应信号 DACK 有效之后即可撤消(参看 8237 的引脚一段)，连续的 DMA 过程仍会进行下去。而在请求传输模式下，8237 每传送一个字节就要检查一下 DREQ，若仍有效则继续传送下一字节，若 DREQ 无效则马上停止传送，退出 DMA。但传输工作现场的地址及字节计数全保存在当前地址寄存器及当前字节数计数器中。待当前进行 DMA 的外设重新准备好，再次发出 DREQ 信号。8237 接着原来的地址和计数值继续进行传输，直至计数结束或外部输入$\overline{EOP}$信号，才停止传送，退出 DMA。请求传输模式在块传输模式上增添一层“查询”的色彩，降低了对外设传输速度的要求.

(4) 级联模式

为了扩展 DMA 通道数，可以把一片主 8237 和几片从 8237 进行级联。每一片从 8237 的

HRQ 和 HLDA 信号接到主片的一对 DREQ 和 DACK 端上，而主片的 HRQ 和 HLDA 信号再接 CPU。这种情况下，各从片的每一通道可作 DMA 传输，而主片除了在从片 和 CPU 之间传递握手信号，还管理各从片的优先级，称为工作在级联方式。

2. 操作类型

无论是字节传输、块传输或请求传输其过程中数据的流向，可以分为三种操作类型。

(1) DMA 读　8237 产生$\overline{MEMR}$和$\overline{IOW}$信号，从选中的存储器中读出数据写入外设。

(2) DMA 写　8237 产生$\overline{IOR}$和$\overline{MEMW}$信号，从外设读出数据写入选中的存储器里。

(3) DMA 校验　若 8237 被编程取这种操作类型，则在 DMA 启动后，外部仍产生时序和地址信号，但所有读写控制信号无效，实际没有数据的传输，不发生任何读写操作，仅仅用来校验电路工作是否正常。

8237 还可以编程实现数据从一个存储区域直接传到另一个存储区域，这要同时用到两个通道：通道 0 作 DMA 读，通道 1 作 DMA 写来共同完成。此过程没有外设参与，因此没有一个外部引入的 DREQ 信号来启动。于是需要对通道 0 写入一个软件 DREQ 请求命令（详见后述），产生 HRQ 信号，启动 DMA 过程。通道 0 用当前地址寄存器的地址到源区存储器中读出数据存入 8237 内部的暂存寄存器（不属于任何一个通道）中，然后通道 1 再将自己当前地址寄存器的值放到地址线上，发出$\overline{MEMW}$信号，把数据从暂存寄存器中写入目的区。这个过程共需要 8 个 S 状态：前 4 个状态为 DMA 读，后 4 个状态为 DMA 写。期间两个通道的地址寄存器都进行修正（加 1 或减 1），通道 1 的字节数计数器控制传输数据块的长度。至所有字节传输完毕，产生$\overline{EOP}$信号。同样也允许外部输入$\overline{EOP}$信号来中止传输。

除对每个通道可以单独编程选择工作模式以外，还可以对整个芯片编程来指定通道共同的功能，例如对优先权及操作时序的选择。这是通过对 8237 的命令寄存器写入命令字来实现的，后面将作详细解释。

3. 内部寄存器的寻址

每片 8237 占用 16 个连续的 I/O 端口地址。在 CPU 用片选信号$\overline{CS}$选中芯片的前提下，由地址信号 $A_3 \sim A_0$ 选择 8237 内部的一个端口，再用$\overline{IOR}$信号或$\overline{IOW}$信号决定是对某个内部寄存器读出还是写入。CPU 对这些寄存器的寻址情况列于表 7-5 中。

从表中可见，前 8 个地址（$A_3=0$）是各个通道单独占有的，每两个地址对应一个通道内部的四个寄存器（计数器）：基地址寄存器和当前地址寄存器用同一个地址同时写入（但只有当前地址寄存器可以读出）；基字节数计数器和当前字节数计数器也是用同一个地址同时写入（但也只有当前字节数计数器可以读出）。由于这四个寄存器（计数器）是 16 位，而 8237 的数据线仅 8 位，所以对它们的读写要分高低字节连续操作两次。8237 内部有一个高/低触发器，每次复位后它处于“0”状态。这时对芯片作一次读写实际上是读写其低字节，同时高/低触发器翻转一次变成“1”状态。下次再作读写就针对高字节，而且高/低触发器又翻转成“0”状态，如此自动重复。用户还可以用下面将要讲到的软件命令使高/低触发器强制清零，以保证对 16 位寄存器的读写是从低字节开始。

表中后 8 个地址（$A_3=1$）是四个通道公用的，主要用以对 8237 写入一些命令（称为软件命令），设定 8237 的某些工作状态。

表 7-5　8237 寄存器的寻址

$A_3A_2A_1A_0$	通道号	读操作($\overline{IOR}$)	写操作($\overline{IOW}$)
0000		读当前地址寄存器	写基(当前)地址寄存器
0001	0	读当前字节数计数器	写基(当前)字节数计数器
0010		读当前地址寄存器	写基(当前)地址寄存器
0011	1	读当前字节数计数器	写基(当前)字节数计数器
0100		读当前地址寄存器	写基(当前)地址寄存器
0101	2	读当前字节数计数器	写基(当前)字节数计数器
0110		读当前地址寄存器	写基(当前)地址寄存器
0111	3	读当前字节数计数器	写基(当前)字节数计数器
1000		读状态寄存器	写命令寄存器
1001		—	写请求寄存器
1010	公	—	写屏蔽寄存器某一位
1011		—	写模式寄存器
1100		—	清除高/低触发器
1101		读暂存寄存器	主清除(软件复位)
1110	共	—	清除屏蔽寄存器
1111		—	写屏蔽寄存器所有位

4．寄存器功能及编程

8237 各寄存器对其工作起着不同的控制作用，在进行 DMA 传输前，必须对各寄存器写入一定的内容，以得到所要求的功能。这就是所谓初始化编程。初始化的内容可分为数值型和功能型两类。

每个通道要把传输中将要访问的存储器的初始地址写入基地址寄存器和当前地址寄存器(同时写入)，还要把要求传输的字节数写入基字节数计数器和当前字节数计数器(也是同时写入)，这就是初始化编程中的数值内容。如前所述，由于这些数值是双字节的，都要从高/低触发器为零状态起，先写入低字节，然后写入高字节。在 DMA 过程中，每传送一个字节，当前地址寄存器的内容要作加 1 或减 1(由模式寄存器而设定)修正，当前字节数计数器要减 1。当前字节数计数器从初始值减到 0，还要再传输一个字节，又从 0 减到 0FFFFH 时，才发出$\overline{EOP}$信号，结束 DMA 过程。因此初始化编程时往字节数计数器中写入的值应比真正传输的字节数减 1。若编程时，在通道的模式寄存器中设置了自动重置功能，当$\overline{EOP}$信号产生时，又自动把基地址寄存器和基字节数计数器的内容再次置入当前地址寄存器和当前字节数计数器中，又可以重复 DMA 传输。

功能编程的内容写入了各功能寄存器中，下面逐一说明。

(1) 命令寄存器

8237 命令寄存器的功能如图 7-18 所示。D_7D_6 两位分别规定了 DACK 和 DREQ 两个信号的有效极性。D_5D_3 两位选择工作时序。D_4 规定优先权编码方式。在存储器到存储器传输时，若 $D_1=1$，则允许将通道 1 指定的目的存储器一批单元的内容全部传成通道 0 指定的源区某一单元内容。若 $D_2=1$ 则禁止本片电路工作。

一片8237只有一个命令寄存器，其内容对四个通道都有效。复位操作后，命令寄存器各位全清零。

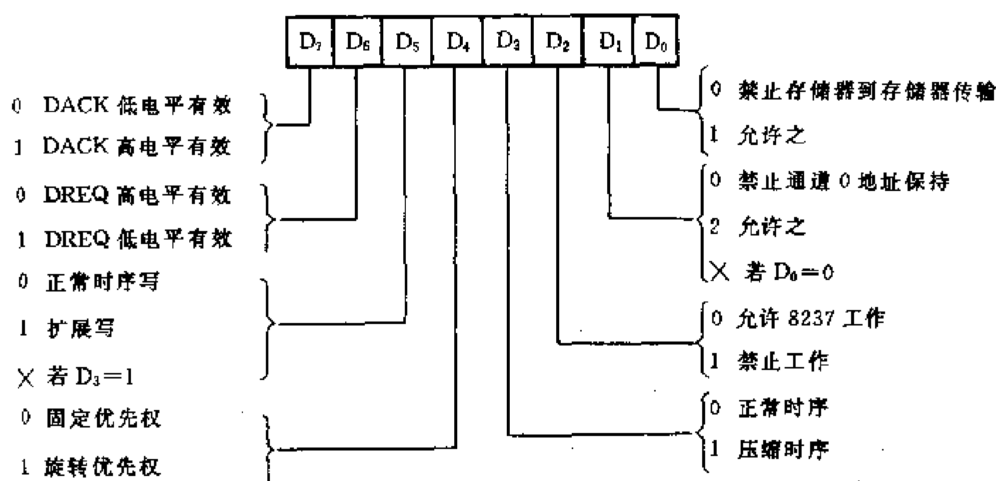


图 7-18　8237的命令寄存器

(2) 模式寄存器

各通道的模式寄存器虽然只有6位，但写入的模式控制字仍是8位，其最低两位用来指定写入的通道号。因此原则上四个通道要写入四个模式字。模式字各位的功能见图7-19。

其中D_7D_6设置了通道的工作模式，D_3D_2规定了其操作类型，D_5指明每传输一个字节当前地址是作加1还是减1。若$D_4=1$，则当一次DMA过程结束，产生$\overline{EOP}$信号时，两个基值寄存器的内容又自动重置到两个当前寄存器中。

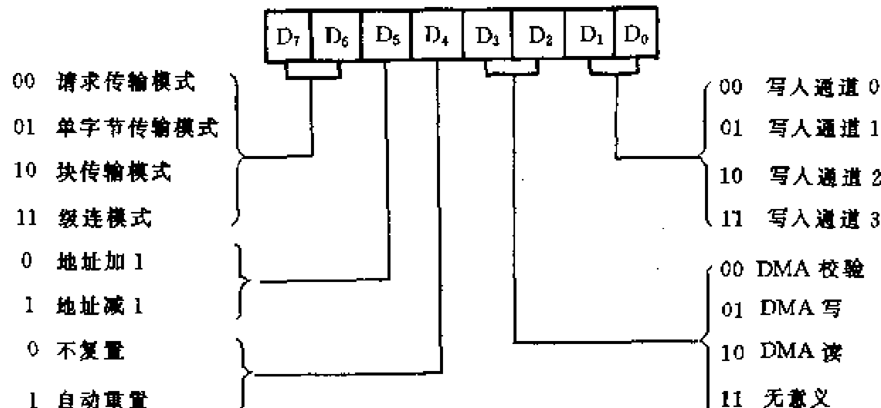


图 7-19　通道模式字

(3) 请求寄存器

每个通道设有一个请求位，可以用软件命令对其进行置位/复位操作。如图7-20所示。对请求位的置位等效于外部产生一个有效的DREQ信号，二者的优先级排队情况也一样。软件请求不受屏蔽寄存器控制，但它只能用于块传送方式。存储器到存储器的传送就只能用通道0的软件请求启动。一个通道的DMA结束时，其请求位被复位。芯片的复位操作清除整个请求寄存器。

(4) 屏蔽寄存器

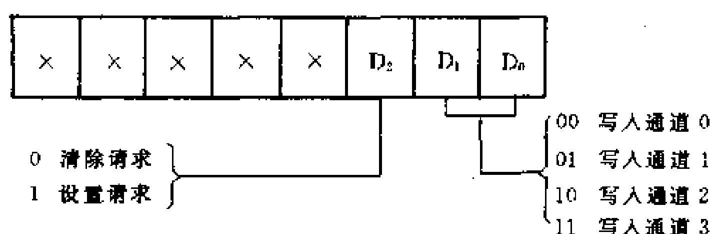


图 7-20 软件请求字

8237 内部有一个公共的屏蔽寄存器，当某位设置为“1”时，其外部对应的 DREQ 信号被屏蔽，不予响应。

表 7-5 中列出可用三个不同的地址对屏蔽寄存器操作：一个是选择地址 $A_3A_2A_1A_0=1010B$，用如图 7-21(a)所示的屏蔽字对某一通道设置或清除屏蔽位。另一种方法是选择地址 $A_3A_2A_1A_0=1111B$，用如图 7-21(b)的屏蔽字同时写四个通道的屏蔽状态。最后一种是选择地址 $A_3A_2A_1A_0=1110B$，对 8237 作一次写操作(虚拟写，不用数据总线)，则将清除屏蔽寄存器所有位(四个通道全开放)。

对芯片的复位操作将屏蔽所有通道。一般情况下，通道在一次 DMA 过程结束后($\overline{EOP}$)，就自动设置屏蔽位，若欲再次传输，必须用软件清除其屏蔽位才行。唯有自动重置方式在每次 DMA 传输结束后不设置屏蔽，以便重置数值后继续传输。

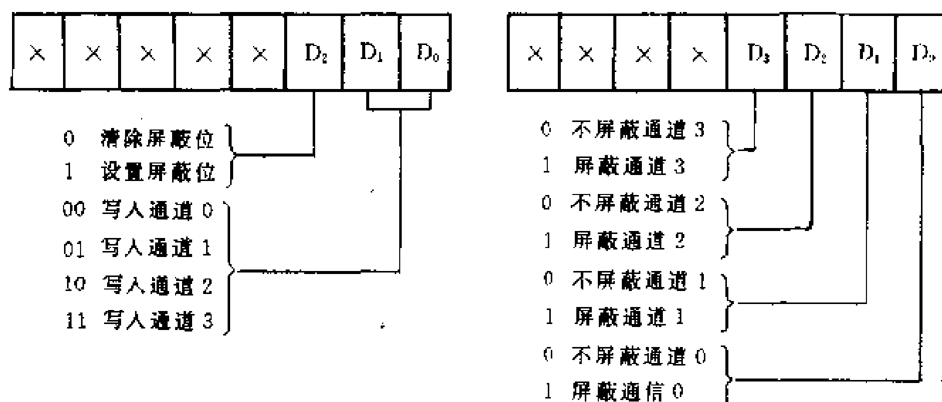


图 7-21 屏蔽字的两种格式

(5) 状态寄存器

在没有进入 DMA 期间，CPU 可以查询 8237 的状态。读入字节的各位为“1”时所表示的含义如图 7-22 所示。其低四位反映各通道传输完成情况。无论是自然计数结束，还是外部输入 $\overline{EOP}$ 信号强制结束，都会使相应位置位。在 CPU 控制总线的情况下，可以对这些位进行查询，以判断传输是否完成。例如在以单字节传输方式传送一批数据的过程中，CPU 会经常获得总线的控制权，通过这种查询来了解 DMA 进行的情况。状态寄存器的高四位记录当前各通道请求的情况，同样为 CPU 查询用。

四、8237 的应用

1. 8237 在系统中的应用

IBM PC/XT 系统板上使用了一片 8237 作 DMA 控制器，其端口地址为 00～0FH。与之配

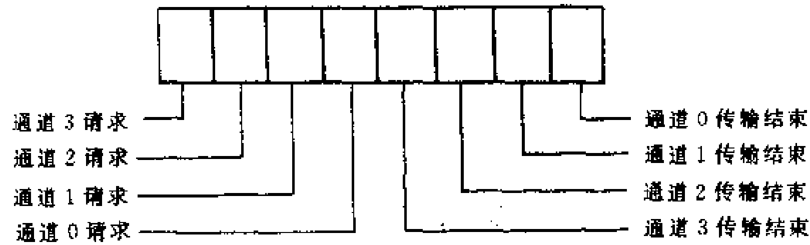


图 7-22 状态寄存器

合工作的还有锁存器，用来锁存数据线 $DB_7 \sim DB_0$ 上分时送出的地址信号 $A_{15} \sim A_8$；还有页面寄存器，口地址为 83H，用来产生地址信号 $A_{19} \sim A_{16}$，这样就得到了完整的 20 位地址信号。

其通道 0 的 $DREQ_0$ 信号接计数定时电路 8253 的 OUT_1 端，大约每隔 15.13μs 产生一次 DMA 请求，用于对动态存储器刷新。

通道 1 为用户保留，$DREQ_1$ 和 $DACK_1$ 信号都引至扩展插槽上。

通道 2 用于软盘驱动器接口。通道 3 用于硬盘接口。$DREQ_2$ 和 $DREQ_3$、$DACK_2$ 和 $DACK_3$ 也都引到扩展插槽上。

四个通道的优先级编码为固定式，通道 0 最高。

通道 0 的每个 DMA 周期包括 $S_1 \sim S_4$ 共四个 S 状态，其它通道都在 S_3 和 S_4 之间插入一个 S_W 等待状态。

系统编程设定 DREQ 信号高电平有效，而 DACK 低电平有效。

2. 应用举例

例 1 现假设用系统板上的 8237 通道 1，将内存起始地址为 80000H 的 300H 字节内容直接输出给外设。其编程可如下所示。

```
MOV  AL,4       ;命令字,禁止 8237 工作
OUT  08,AL      ;写命令寄存器
MOV  AL,0       ;
OUT  0DH,AL     ;主清除,清除高/低触发器
OUT  02,AL      ;写低位地址 00
OUT  02,AL      ;写高位地址 00
MOV  AL,8       ;页面地址为 8
OUT  83H,AL     ;写入页面寄存器
MOV  AX,300H    ;传输字节数
DEC  AX
OUT  03,AL      ;写字节数低位
MOV  AL,AH      ;
OUT  03,AL      ;写字节数高位
MOV  AL,49H     ;模式字;单字节读,地址加 1
OUT  0BH,AL
MOV  AL,44H     ;命令字;DACK 和 DREQ 低有效
OUT  08H,AL     ;正常时序,固定优先权
```

```
MOV   AL,01        ;清除通道 1 屏蔽
OUT   0AH,AL
WAITFIN AL,08      ;读通道 1 状态
AND   AL,02        ;传输完成否
JZ   WAITF         ;没完成则等待
MOV   AL,05        ;完成后屏蔽通道 1
OUT   0A,AL        ;
⋮
```

下面我们再举一个简单的例子，以阐明 8237 的工作。

例 2　图 7-23 是某 8 位 A/D 转换器的简化引线图以及它的工作时序。图中，利用 A、B、C 的不同编码可以选择不同输入端的模拟信号。从编码 000 到 111 依次选择 IN0 到 IN7。START 是启动变换信号，EOC 是变换结束信号，EN 为 A/D 变换器允许输出信号，当 EN=1 时，才能打开 A/D 变换器内部的三态门，8 位数据 $D_0 \sim D_7$ 才能出现在数据总线上。当 EN=0 时，A/D 变换器 $D_0 \sim D_7$ 呈现高阻状态。

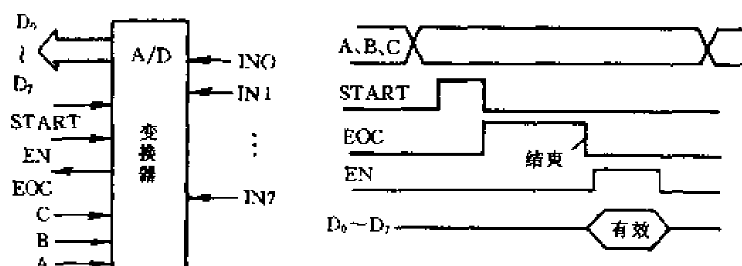


图 7-23　8 位 A/D 变换器及工作时序图

我们准备利用 DMA 方式，来获得 A/D 变换器的数据。A/D 变换器在这种方式下的连接简图如图 7-24 所示。

```
OUT   DMA+4,AL          ;写入通道 2
MOV   AL,MENH           ;存储口地址高字节
OUT   DAM+4,AL;
MOV   AL,8
OUT   DAM+5,AL          ;计数低字节
MOV   AL,0
OUT   DAM+5,AL          ;计数高字节
MOV   AL,PAGE           ;页寄存器装入数据
OUT   APGF,AL           ;
MOV   AL,56H            ;方式字写入通道 2：写传送
OUT   DAM+11,AL         ;自动予置，地址递增，单字节方式
MOV   AL,0
OUT   DAM+1, AL         ;写入命令，允许 8237 工作，正常时序 DREQ 高有
                         效，DACK 低有效
```

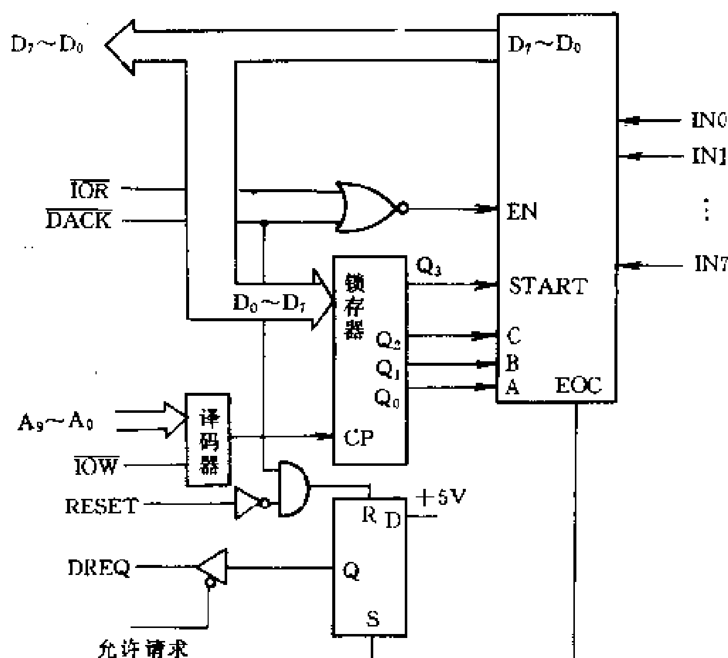


图 7-24 8 位 A/D 转换传送电路框图

```
MOV   AL,0
OUT   DAM+15,AL          ;写屏蔽寄存器
```

在初始化 8237 时，有些初始化字是为各通道工作所独有的，而有的初始化字是各通道所共有。共有的初始化字可以在初始化任何一个通道时加以初始化。

思考题与习题

1. 输入/输出设备有哪些特点？CPU 通过什么与输入/输出设备通信？

2. CPU 与输入/输出设备通信时所用到的接口电路通常应具备的功能？

3. 计算机输入/输出方式有哪几种？它们各有什么特点？

4. 何谓"独立的 I/O 端口编址方式"？何谓"与存储器统一的 I/O 编址方式"？这两种编址方式各有什么优缺点？

5. 在 CPU 与外设备交换信息的过程中，直接程序控制传送方式、程序中断方式、DMA 方式各自的适应范围是什么？下面这些结论正确吗？为什么？

(1)程序中断方式能提高 CPU 的利用率，所以在设置了中断方式后就不必再应用直接程序控制方式了。

(2)DMA 能处理高速外设与主存间的数据传送，高速工作性能往往能覆盖低速工作需要。所以 DMA 方式完全可以取代中断方式。

6. 利用三态门(74LS244)作为输入接口，接口地址规定为 04E5H，试画出其与 8086 总线的连接图。

7. 利用具有三态输出的锁存器(74LS374)作为输入接口，接口地址为 0E504H，试画出连接图。若上题中输入接口的 bit_3、bit_4 和 bit_7 同时为 1 时，将 DATA 为首地址的十个内存数据连续由输出接口输出；若不满足条件则等待，试编程序。

8. 选择题：

(1) 通常，在 I/O 接口自身都有定时控制逻辑，其原因是它与主机交换信息时存在( )。

A. 数据需要缓冲　　B. 速度不匹配

C. 时序不同步　　D. 数据格式需转换

(2) 某台微型机采用 I/O 端口独立编址，分配的端口号共 1024 个，处理机用 I/O 指令访问的寄存器可多达( ).

A. 102　B. 512　C. 2048　D. 1023

(3) 某台微机采用 I/O 端口统一编址，处理机可用一条 ADD 加法指令对( )访问。

A. 整个地址空间　　B. RAM 地址空间

C. I/O 地址空间　　D. 随机器设计而定

(4) 程序查询流程总是按( )次序完成一个字符的传输。

A. 读状态端口，写数据端口

B. 写数据端口，读状态端口，写数据端口

C. 写控制端口，读状态端口，写数据端口

D. 随 I/O 接口的具体要求而定

(5) 8086/8088CPU 响应硬件中断 INTR 请求的必要条件除 IF=1 外，还需满足( )

A. 访存操作结束　　B. 当前指令执行完

C. 无软件中断请求　　D. 无内部中断请求

(6) CPU 对 DMA 控制器提出的总线请求响应要比中断请求的响应快，其原因是( )

A. 只需完成访内存操作　　B. 只需释放总线控制权

C. 无需保留断点现场　　D. 有硬件 DMA 控制器

(7) DMA 传送结束由 I/O 接口向 CPU 发出中断请求，其目的是( )

A. 让 CPU 收回总线控制权　　B. 让 DMA 控制器释放总线控制

C. 让 CPU 检查 DMA 操作正确性 D. 让 DMA 控制器复位，准备下一次 DMA 传输

(8) PC 系列的总线控制器受控于微处理机的操作状态，当 CPU 在取指状态下，它接收的状态值和相应输出的控制信号分别是( )

A. 101 和 MEMR　　B. 110 和 MEMR

C. 100 和 MEMR　　D. 100 和 MEMR

9. 试画一个流程图，说明如何用程序控制的 I/O 把 N 个字节的数据块输入到存储器中。

10. 若图 7-24 为开关闭合个数指示接口电路，用于显示闭合开关个数。试编写程序段统计闭合开关个数并显示于显示器上。

11. 编写一个轮流测试两台设备的状态寄存器的程序，若状态寄存器的第 0 位为 1，则从相应设备输入一个字节的数据，如果任一个状态寄存器的第 3 位是 1，则停止输入过程。状态寄存器的地址为 0024H 和 0036H，相应的数据输入缓冲寄存器的地址为 0026H 和 0038H。输入的数据假定被存入从 $BUFF_1$ 和 $BUFF_2$ 开始的存储缓冲区中，$BUFF_1$ 和 $BUFF_2$ 已定义为字节变量。

12. 图 7-25 所示为开关量检测与显示电路接口。显示器采用 7 段 LED 显示器，由 BCD-7 段译码/驱动器所驱动，并采用共阴极接法(只有阴极为低电平，显示器才会发亮)，假定任何时候至多只有一只开关闭合，试编写一程序，显示闭合开关序号。若无开关闭合，则显示器不发亮。输入缓冲器和输出数据锁存器地址都为 20H。

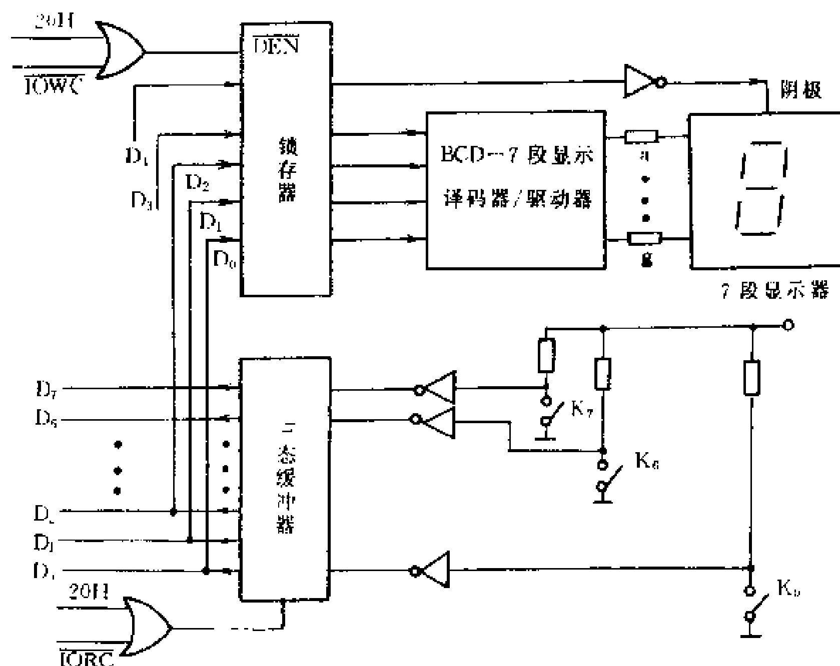


图 7-25

13. 图 7-26 为 7 段显示器接口，显示器采用共阳极接法，试编写程序段，使 AL 中的一位十六进制数（AL 的高位为 0000）显示于显示器上，输出锁存器地址为 20H。

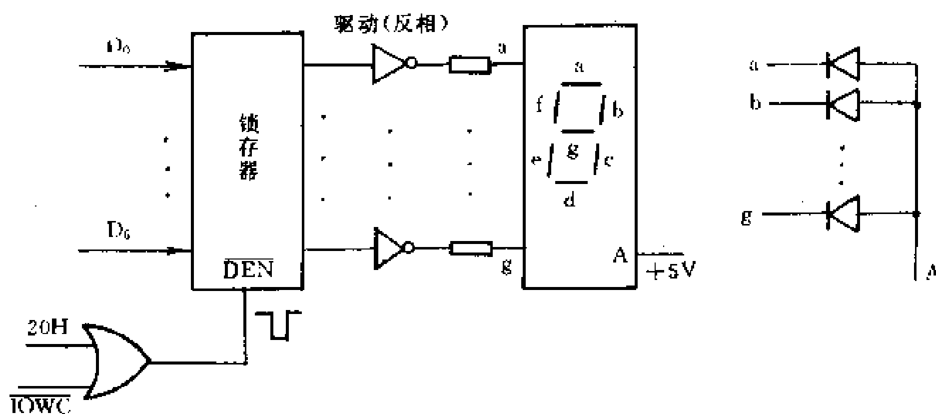


图 7-26

14. 图 7-27 为控制台面板按键接口，编写查询按键状态的程序段，当某个按键闭合时，调用相应的过程完成处理操作任务，各过程的名字如下：

按键	过程名
K0	TEST-PROC
K1	PRINT-PROC
K2	FETCH-PROC
K3	STORE-PROC
K4	RUN-PROC
K5	SEND-PROC
K6	COMM-PROC

K7 POWER-PROC

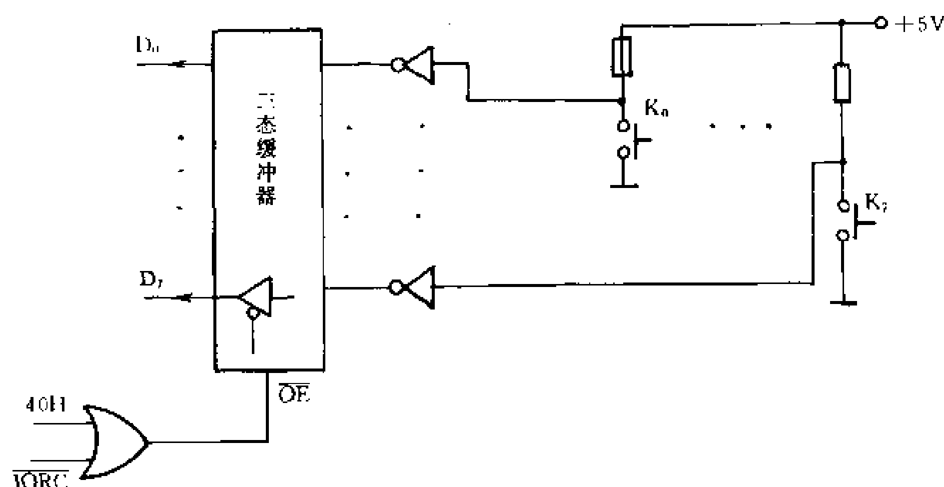


图 7-27

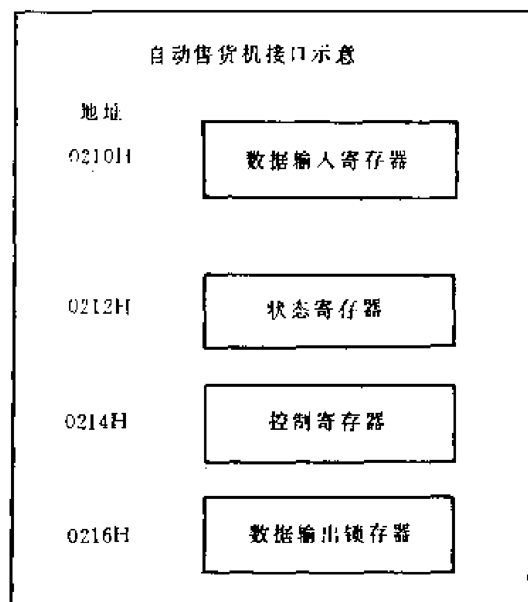


图 7-28

15. 图 7-28 所示为自动售货机接口示意图，购货者必须投入相当于 50 分的硬币后，再揿一次取物按纽，若投入的硬币为 50 分，则售货机将一袋食品送回。该接口是这样设计的，每投入一枚硬币，状态寄存器的第 0 位置 1，并使数据输入寄存器中存入该硬币的代码（10 分的代码为 01，25 分为 02，50 分为 03），当数据输入寄存器内容被读入时，状态寄存器的 0 位被清除，并从数据输出锁存器（它控制显示器）输出已投入的硬币的总数，输出数据形式为 BCD 码（如投入一枚 25 分硬币，输出 25H，再投入一枚 10 分硬币，输出 35H......）。当取物按钮被揿下时，状态寄存器的第 1 位置 1。若投入的硬币总额为 50 分，控制寄存器的位 0 为 1，启动送货机构，并取入硬币，若总额不是 50 分，使控制器的位 7 置 1，以控制退出所投入的全部硬币。无论投入的硬币总额为多少，一旦使控制寄存器的位 0 或位 7 置位，就使显示器显示 00。试编写上述接口的相应软件。

16. CPU 响应 DMA 请求和响应中断请求有什么本质性的区别?

17. DMAC(8237)占几个端口地址? 这些地址在读写时的作用是什么? 叙述 DMAC 由内存向端口传送一个数据块的过程。若希望利用 8237 把内存中的一个数据块传送到内存的另一个区域，应当如何处理? 当考虑 8237 工作在 8086 系统，数据是由内存的某一段向另一段传送且数据块长度大于 64K 时，应当如何考虑?

第八章　中　断

第一节　中断原理

一、中断过程

当外部设备准备好与CPU传送数据，或者有某些紧急情况需要处理，也许是定时时间到等等。这时，外设向CPU发出中断请求，CPU接收到请求并在一定条件下，暂时停止执行原来的程序而转去中断处理，处理好中断服务再返回继续执行原来程序，这就是一个中断过程。下面我们将中断过程分成三个阶段来讨论。

1. 中断请求

中断是CPU被动地响应外设要求的服务，发出中断请求的外部设备称为中断源，一般中断源有：

(1)数据输入/输出外设请求中断；

(2)定时时间到申请中断；

(3)满足规定状态申请中断；

(4)电源掉电申请中断；

(5)故障报警申请中断；

(6)程序调试设置中断等。

2. 中断响应

CPU在没有接到中断请求信号时，一直执行原来的程序(称为主程序)。由于外设的中断申请随机发生，有中断申请后CPU能否马上为其服务呢?这就要看中断的类型，若为非屏蔽中断申请，则CPU执行完现行指令后，做好保护现场工作即可去处理中断服务；若为可屏蔽中断申请CPU只有得到允许才能去服务。这就是说，CPU能否在接到中断申请后立即响应要视情况而定。对可屏蔽的中断申请，CPU要响应，必须满足以下三个条件：

(1)无总线请求；

(2)CPU被允许中断；

(3)CPU执行完现行指令。

CPU响应中断要自动完成三项任务：

(1)关闭中断；

(2)保护关键现场，通常为断点和标志寄存器内容入栈；

(3)获得中断服务程序入口地址，转中断服务程序。

3. 中断处理

一旦CPU响应中断，就可转入中断服务程序之中，中断处理要做以下六件事：

(1)保护现场

CPU响应中断时自动完成断点和标志寄存器内容的保护，但主程序中使用的寄存器的

保护则由用户视使用情况而定。由于中断服务程序中也要用到某些寄存器，若不保护这些寄存器在中断前的内容，中断服务程序会将其修改，这样，从中断服务程序返回主程序后，程序不能正确执行，由用户保护这些寄存器内容的功能称为保护现场，实质上是执行 PUSH 指令将需要保护的寄存器的内容推入堆栈。

(2)开中断

CPU 接收并响应一个中断后自动关闭中断，是为了不允许其他的中断来打断它。但在某些情况下，有比该中断更优先的情况要处理，此时，应停止对该中断的服务而转入优先级更高的中断处理，故需要再开中断，若不允许响应更高级别的中断请求在此也可不开中断。

(3)中断服务

中断服务的核心就是对某些中断情况的处理。如：传送数据，处理掉电紧急保护，各种报警状态的控制处理等等。

(4)关中断

由于有(2)的开中断，因而在此处对应一个关中断过程，以便下面恢复现场的工作顺利进行而不被中断。

(5)恢复现场

在返回主程序前要将用户保护的寄存器内容从堆栈中弹出，以便返主后继续正确执行主程序，恢复现场用 POP 指令。要注意的是堆栈为先进后出的数据结构，注意保护现场时寄存器入栈的先后次序要与出栈时的次序相反。

(6)开中断返回

此处的开中断对应 CPU 响应中断后自动关闭中断，在返回主程序前，也就是中断服务程序的倒数第二条指令往往是开中断指令，最后一条是返回指令，执行返回指令，CPU 自动从现行堆栈中弹出 CS、IP 和 Flags 的内容，以便继续执行主程序。

中断过程如图 8-1 所示。

图 8-1　中断过程示意图

二、中断源的识别

一个微机系统往往接有多个外部设备，当外设与 CPU 以中断方式进行通讯时，CPU 必须从这多个外设中判别出正在申请中断的设备，找到相应的服务程序首地址，才能转去为其服务。因而要解决的问题是：确定中断源、找到该中断服务程序的首地址。

识别中断源方法通常有两种：

1. 查询中断(软件识别)

查询中断是用软件查询的方法识别中断源。在外设有中断请求的前提下，从多个设备中查

找请求中断的那个设备。

一般 CPU 的中断请求信号线较少，而发出中断申请的设备较多，此时，要设置一个中断查询接口电路，用以锁存中断申请信号并提供给 CPU 查询。

查询中断利用一段查询程序，不但可以确定请求服务的设备，同时还可以转至相应的服务程序。当然，服务程序是预先编好存放在内存区的，可以存放在内存的任意可存放的区域之中。

2. 矢量中断(硬件识别)

对中断源识别最快的方法是矢量中断，该方法要求外设提供中断申请信号和中断矢量标志，进一步获得中断服务程序的入口地址，又称中断矢量。每个外设都预先约定好各自的中断矢量。当 CPU 响应某个外设的中断请求时，控制逻辑就将该外设的中断矢量送入 CPU，以自动指向相应的中断程序的入口地址，转入中断服务。8086CPU 就是以矢量中断的方法实现中断源的识别的。

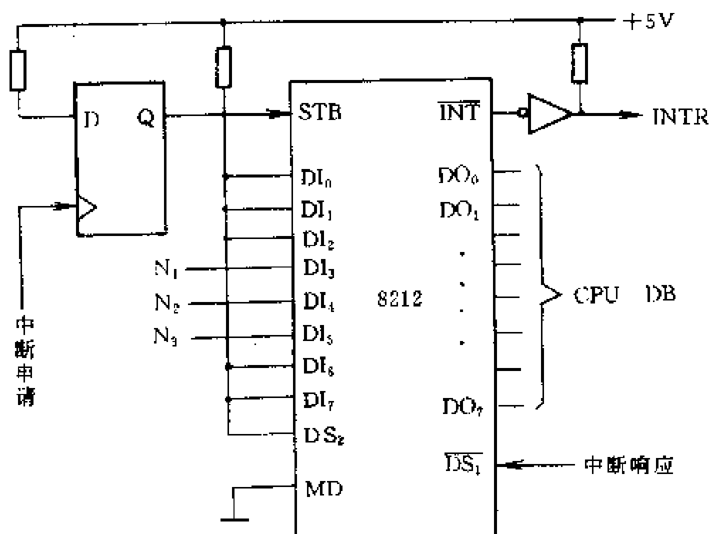


图 8-2 8212 作为矢量中断的接口图

中断矢量可以由专用的中断控制芯片提供，也可由接口芯片 8212 提供。图 8-2 是 8212 芯片做为矢量中断的接口电路图。当外设有中断申请，则使 D 触发器输出高电平提供给 8212，当 8212 的 STB 信号为高电平，则$\overline{INT}$信号有效，通过非门可向 CPU 发出中断请求信号 INTR。当 CPU 响应中断，使$\overline{DS_1}$有效，故中断矢量从 8212 的 DO_0～DO_7 送 CPU 的数据总线。中断矢量的值由 DI_0～DI_7 八个输入端的状态决定，图 8-2 中，DI_7、DI_6 以及 DI_2、DI_1、DI_0 均为“1”，DI_5、DI_4、DI_3 也接高电平“1”，则矢量标志为 FFH，也可选择其他值。当 8212 要为多个中断源提供中断矢量时，可用 N_1、N_2 及 N_3 的组合状态提供。

三、中断优先级的确定

在一个微机系统中，常常遇到多个中断源同时申请中断的情况。这时，CPU 必须确定首先为哪一个中断源服务，以及服务的顺序。另外，有些中断服务是不允许其他中断打断的，有些中断要优先处理，另外有些中断，在服务期间可以接受比它更需要紧急处理的中断等等。解决这些问题就是解决中断的优先排队问题。有两种方法解决中断优先级的问题。

1. 软件查询

中断优先级由查询顺序决定，先被查询的中断源具有高的优先级。使用这种方法需要设置一个中断请求信号的锁存器，将各中断源的请求信号保存下来，以便查询并可对还没有服务的中断请求作一个备忘录。软件查询的好处在于可以用修改软件来改变中断优先级，而不必要更改硬件。软件查询确定优先级的缺点是，响应中断慢，服务效率低，因为优先级最低的中断源申请的服务，必须先将优先级高的设备查询一遍，若设备较多，有可能优先级低的中断源很难

得到服务。

2. 硬件处理

(1)编码器组成中断优先级电路

74LS148 是一个 8～3 优先级编码器，它是一个 16 管脚双列直插式 TTL 器件。其引脚及功能真值表如图 8-3

74LS148

信号	引脚	引脚	信号
I_4	1	16	V_{CC}
I_5	2	15	E_0
I_6	3	14	G_S
I_7	4	13	I_3
E_I	5	12	I_2
A_2	6	11	I_1
A_1	7	10	I_0
GND	8	9	A_0

输入									输出				
E_I	0	1	2	3	4	5	6	7	A_2	A_1	A_0	G_S	E_0
1	×	×	×	×	×	×	×	×	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	×	×	×	×	×	×	×	0	0	0	0	0	1
0	×	×	×	×	×	×	0	1	0	0	1	0	1
0	×	×	×	×	×	0	1	1	0	1	0	0	1
0	×	×	×	×	0	1	1	1	0	1	1	0	1
0	×	×	×	0	1	1	1	1	1	0	0	0	1
0	×	×	0	1	1	1	1	1	1	0	1	0	1
0	×	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1

图 8-3　74LS148 编码器引脚图及真值表

74LS148 编码器有 I_0～I_7 八个输入引脚，可接收来自八个中断源的申请信号（低电平的有效信号），E_I 为低电平有效的片选输入信号。从真值表中可知第 I_7 号输入引脚上的中断请求具有最高优先级，只要第 I_7 号输入引脚上为“0”，则输出引脚 $A_2A_1A_0$ 为 000，I_0 号引脚上的中断具有最低优先级，其 $A_2A_1A_0$ 的输出编码为 111。

(2)链式优先级排队

这种方法是利用外设在系统中的物理位置来决定其中断优先级的，电路如图 8-4。图中，

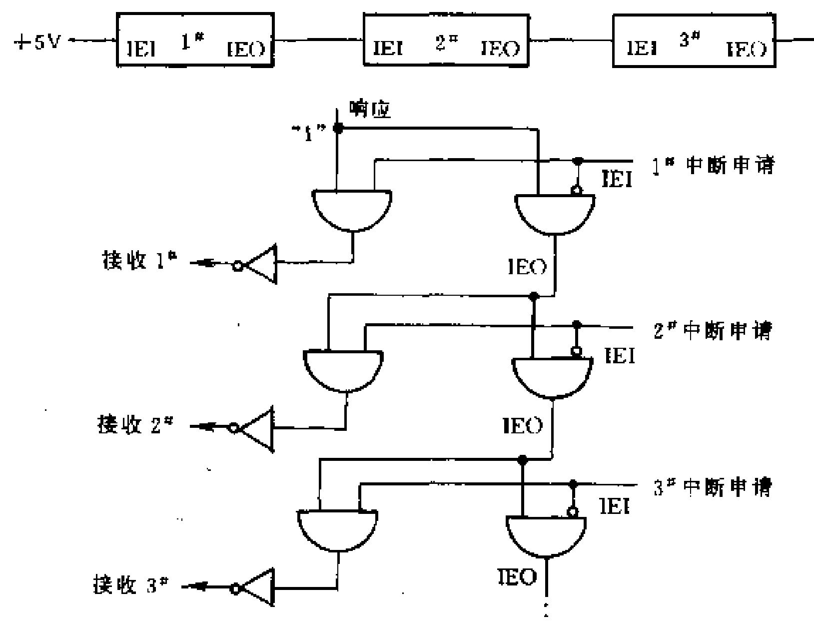


图 8-4　链式中断优先级电路

若 1＃设备发出中断请求（高电平信号），且 CPU 响应时，1＃设备的申请被接收，同时封锁 2

#,3#的中断请求。当 1# 设备无中断请求，而 2#设备有中断请求时，CPU“响应”信号通过第一级 IEO 传递给 2#中断申请的“与”门，使 2#中断申请被接收，同时封锁 3#中断请求。在响应 2#中断并为其服务期间，1#设备发出中断申请，则 CPU 会挂起 2#的服务转去接收优先级高的 1#设备的申请并为其服务。1# 服务完毕，再继续为 2#设备服务。显然，链式优先级排队电路使优先级别高的设备的中断不被优先级别低的设备所打断，但可随时中断优先级别低的服务。在大部分可编程的 I/O 接口芯片中，都有中断优先链，从引脚上体现为链的输入 IEI 和链的输出 IEO。

第二节　8086 中断系统

一、8086 中断类型

8086/8088 中断属矢量中断也叫类型中断，共有 0～255 种类型中断，可分为软件中断和硬件中断。

1. 软件中断

由 CPU 执行某些指令引起的中断称为软件中断(亦称内部中断)。软件中断包括以下情况：

(1)除法出错中断

在 CPU 作除法运算时，若除数为零或商超出了有关寄存器所能表示的数值范围，则产生除法出错中断，其类型为 0。

(2)单步中断

在标志寄存器 FLAGS 中的跟踪标志 TF=1 且中断允许标志 IF=1 时，每执行一条指令就引起一次中断。在单步调试程序时使用，单步中断的类型号是 1。

(3)INTO 溢出中断

当溢出标志 OF=1 时，执行指令 INTO，则产生溢出中断。两个条件中任何一个不具备，溢出中断则不发生。溢出中断的类型为 4。

(4)中断指令 INT　n

在 8086/8088 指令系统中有一类中断指令 INT n，其中 n 为中断类型号(0～255)。CPU 执行一条这种指令，发生一次中断。在 IBM PC/XT 的操作系统中，用不同类型号编入一些标准的中断服务程序，用户程序可以用 INT n 指令方便地调用，也可以用此调试中断服务程序。

2. 硬件中断

由 CPU 外部中断请求引脚 NMI 和 INTR 引起的中断称为硬件中断(亦称外部中断)。硬件中断又分为非屏蔽中断和可屏蔽中断两种。

(1)非屏蔽中断

若是 CPU 的 NMI 引脚接收到一个正跳变信号，则可能产生一次非屏蔽中断。这种中断的响应不受中断允许标志 IF 的控制。8086/8088 要求 NMI 信号跳变成高电平后至少保持两个时钟周期以上的宽度，以便锁存下来，待当前指令完成之后响应。

IBM PC/XT 中的非屏蔽中断源有三种：浮点运算协处理器 8087 的中断请求，系统板上 RAM 的奇偶校验错及扩展槽中的 I/O 通道错。以上三者中任何一个都可以单独提出中断请求，但是否真正形成 NMI 信号，还要受 NMI 屏蔽寄存器的控制。当这个寄存器的 $D_7=1$ 时才

允许向 CPU 发 NMI 请求，否则即使有中断要求，也不能发出 NMI 信号。NMI 屏蔽寄存器的口地址为 A0H，可以用输出指令对这一位写入“1”或者“0”达到允许或禁止 NMI 的效果。

NMI 被响应时，自动产生中断类型号 2 中断，并转入相应服务程序。

(2)可屏蔽中断

若是一个高电平信号加到 CPU 的 INTR 引脚，且中断允许标志 IF=1，则产生一次可屏蔽中断。当 IF=0 时，INTR 的中断请求被屏蔽，在 IBM PC/XT 中，所有可屏蔽的中断源都先经过中断控制器 8259A 管理之后再向 CPU 发出 INTR 请求。

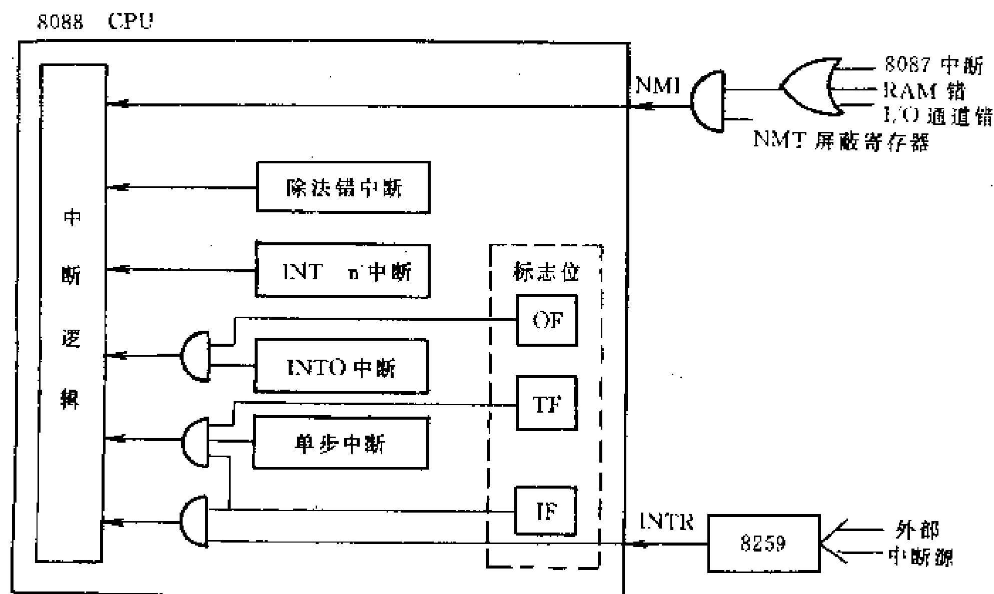


图 8-5 IBM PC/XT 的中断类型

二、8086 的中断处理

1、中断管理

8086 系统把中断服务入口地址表中的中断明确地分成三部分。第一部分是类型 0 到类型 4，共 5 种类型，定义为专用中断，它们占表中 000～013H，共 20 个字节。这 5 种类型中断的入口已由系统定义，不允许用户作任何修改。其中：

INT0——除法出错中断

INT1——单步中断

INT2——外部引入不可屏蔽中断

INT3——断点中断

INT4——或 INTO——溢出中断

第二部分是类型 5 到类型 31H 为系统备用中断，这是 Intel 公司为软、硬件开发保留的中断类型，一般不允许用户改作其他用途，其中许多中断已被系统开发使用，例如类型 21H 已用作系统功能调用的软件中断。

第三部分是类型 32H 到 0FFH，可供用户使用。这些中断可由用户定义为软中断，由 INT 指令引入，也可以是通过 INTR 引脚直接引入的或通过中断控制器 8259A 引入的可屏蔽硬件中断。

中断服务入口地址表又可称为中断指针表或中断矢量表，每个入口都是低位字为偏移地

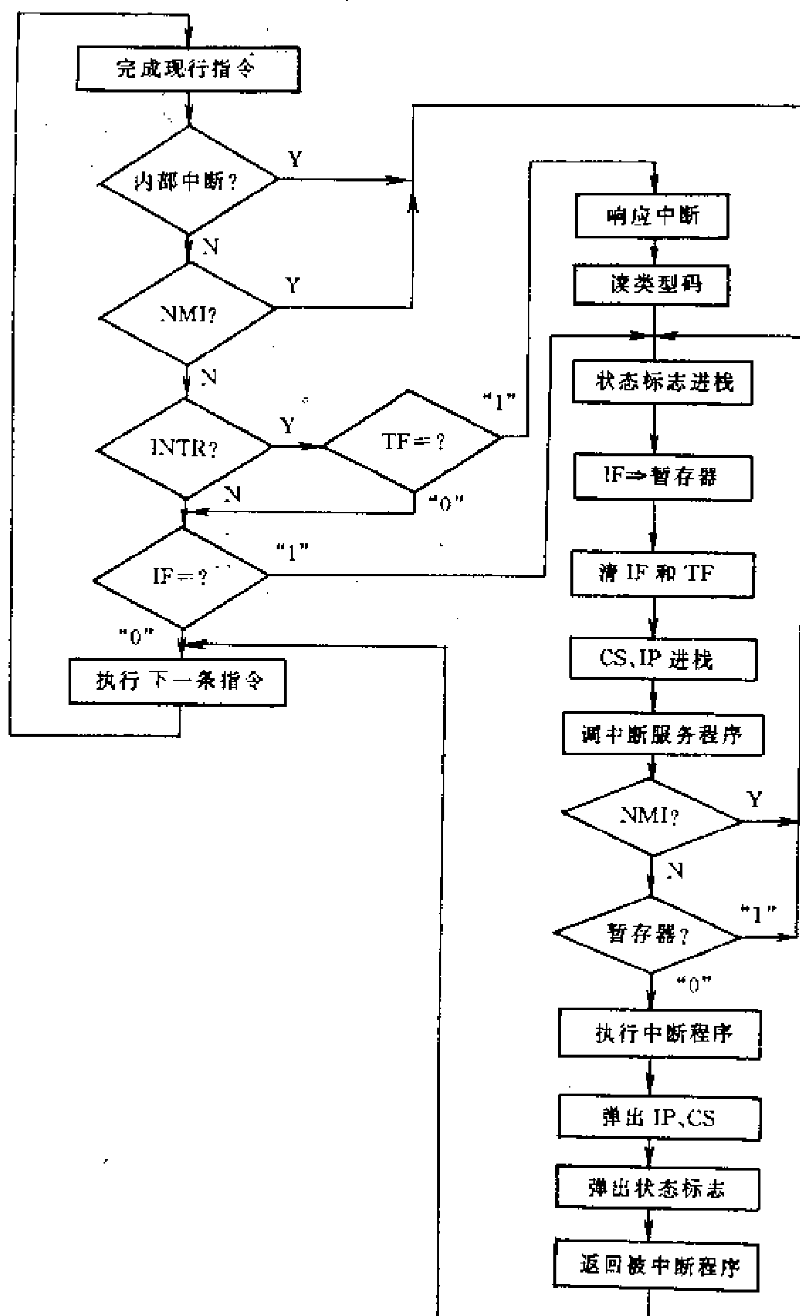


图 8-6　8086 中断处理顺序流程图

址，高位字为段基值。如表 8-1 所示。

2、中断处理顺序

8086CPU 的中断优先级序列从高到低为：

(1)除法出错中断，溢出中断，INT　n

(2)NMI

(3)INTR

(4)单步中断

8086CPU 对中断的处理可用图 8-6 的流程表示。

3、中断类型号的获取

有两种方法获取类型号，第一种是用直接获取。对于类型号 0～4 的中断，由于 8086CPU

表8-1　中断服务入口地址表

	地址	入口
专用中断	000H	除法中断入口
	004H	单步中断入口
	0084H	NMI中断入口
	00CH	断点中断入口
	010H	溢出中断入口
	014H	类型5中断入口
		⋮
	07FH	类型31中断入口
	080H	类型32断入口
		⋮
	3FCH	类型255中断入口

已规定了产生中断的原因，只要有相应中断就可获得相应类型号，同理，许多系统调用功能是用 INT n 指令直接获取类型号的。第二种是由外部引入的 INTR 中断，这类中断必须由硬件提供中断类型号，当 CPU 在响应中断的响应周期，进行到第二个 $\overline{\text{INTA}}$ 周期时，用 $\overline{\text{INTA}}$ 将类型号放入数据总线，CPU 从数据总线上获取类型号，并自动将类型号乘 4，作为地址指针，获得中断服务程序的入口地址，转入相应服务程序。图 8-2 給出由 8212 芯片产生中断类型号的电路。

当外设申请中断，通过 8212 向 CPU 发出中断请求，若满足条件，CPU 响应中断，则回答 $\overline{\text{INTA}}$ 信号，该低电平有效信号选通 8212，中断请求被响应，执行第二个总线周期时，利用 $\overline{\text{INTA}}$ 信号，将中断类型号取入 CPU，并乘 4 得到矢量表的地址，从该地址的存储单元中取出该中断服务程序的段基值及偏移地址，这样 CPU 就可自动转入中断服务程序执行。

表 8-2 中列出 PC 机中主要的中断及它们的中断类型号。

表 8-2　PC 机中主要的中断及它们的中断类型号的分配表

类型号（十六进制）	服务	类型号	服务
0	除法出错	1AH	软件时间服务
1	单步	1BH	产生 Ctrl Break 中断
2	不可屏蔽中断	1CH	每个时钟信号中断
3	断点中断	1DH	视频控制参数表
4	溢出中断	1EH	软盘驱动参数表
5	打印屏幕	1FH	视频图形字符
		20H	程序结束服务
		21H	DOS 所有函数调用
8	IR_0（主体）	22H	结束子程序地址
9	IR_1	23H	键盘处理地址
0AH	IR_2	24H	致命错误地址
0BH	IR_3	25H	绝对读磁盘
0CH	IR_4	26H	绝对写磁盘
0DH	IR_5	27H	终止程序但驻留内存
0EH	IR_6	2FH	DOS 多路中断
0FH	IR_7	70H	IR_8（从片）
10H	显示器	71H	IR_9

（续）

类型号（十六进制）	服务	类型号	服务
11H	取外设列表	72H	IR_{10}
12H	取基本内存容量	73H	IR_{11}
13H	硬磁盘	74H	IR_{12}
14H	串行口	75H	IR_{13}
15H	系统服务	76H	IR_{14}
16H	键盘	77H	IR_{15}
17H	打印机		
18H	切换到 BASIC 控制		
19H	重新启动		

三、80386/80486 的中断

1. 中断和异常

80386/80486 不仅具有前面讲到的所有的中断类型，而且大大丰富了内部中断的功能，把许多执行指令过程中产生的错误情况也纳入了中断处理的范围，这类中断称为异常中断，简称异常(Exceptions)。从总体上异常分为三类：失效(Faults)，陷井（Traps）和中止(Abort)。三类异常的差别表现在两方面：一是发生异常的报告方式，二是异常中断服务程序的返回方式。

(1)失效：若某条指令在启动之后真正执行之前被检测到异常，即产生异常中断，而且在中断服务完成后返回该条指令，重新启动并执行完成。例如在读虚拟存储器时，首先产生存储器页失效或段失效，其中断服务程序立即按被访问的页或段将虚拟存储器的内容从磁盘上转移到物理内存中，然后再返回主程序中重新执行这条指令，于是可以正常执行下去。

(2)陷阱：产生陷阱的指令在执行后才被报告，且其中断服务程序完成后返回到主程序中的下一条指令。例如用户自定义的中断指令 INT n 就属于此类型。

(3)中止：异常发生后无法确定造成异常指令的实际位置，例如硬件错误或系统表格中的错误值造成的异常。在此情况下原来的程序已无法继续执行，因此中断服务程序往往重新启动操作系统并重建系统表格。

2. 保留的中断

80386/80486 最多可以定义 256 个不同的中断或异常。其中系统已经定义了一些保留的中断及异常，剩下的可供用户自行定义。

3. 中断描述表

为了管理各种中断，80386/80486 设立了一个中断描述表 IDT（Interrupt Descriptor Table)。表中最多可包含 256 个描述项，对应 256 个中断或异常。描述项中包含了各个中断服务程序入口地址的信息。

当 80386/80486 工作于实地址方式时，系统的 IDT 变为 8086/8088 系统中的中断矢量表，存于系统物理存储器的最低地址区中，共 1K 字节。每个中断矢量占 4 字节，即 2 字节的 CS 值和 2 字节的 IP 值。

当 80386/80486 工作于保护方式时，系统的 IDT 可以置于内存的任意区域，其起始地址存放在 CPU 内部的 IDT 基址寄存器中。有了这个起始地址，再根据中断或异常的类型码，即可取到相应的描述项。每个描述项占 8 个字节，包括 2 字节的选择器，4 字节的偏移量，这 6 字节共同决定了中断服务程序的入口地址，其余 2 字节存放类型值等说明信息。

第三节　可编程中断控制器 8259A

8259A 可编程中断控制器(PIC)，又称优先级控制器，它可为 CPU 管理和处理八级矢量优先级中断，具有单＋5V 电源供电，可与其他 8259A 芯片级联来扩大中断功能，优先级方式在执行主程序的任何时间里能够动态地改变，无论 8086CPU 工作在最小模式还是最大模式，都可以与之配套使用。

一、8259A 的内部结构及引脚

1. 内部结构

8259A 的内部结构框图如图 8-7 所示，它由以下几个部分组成：

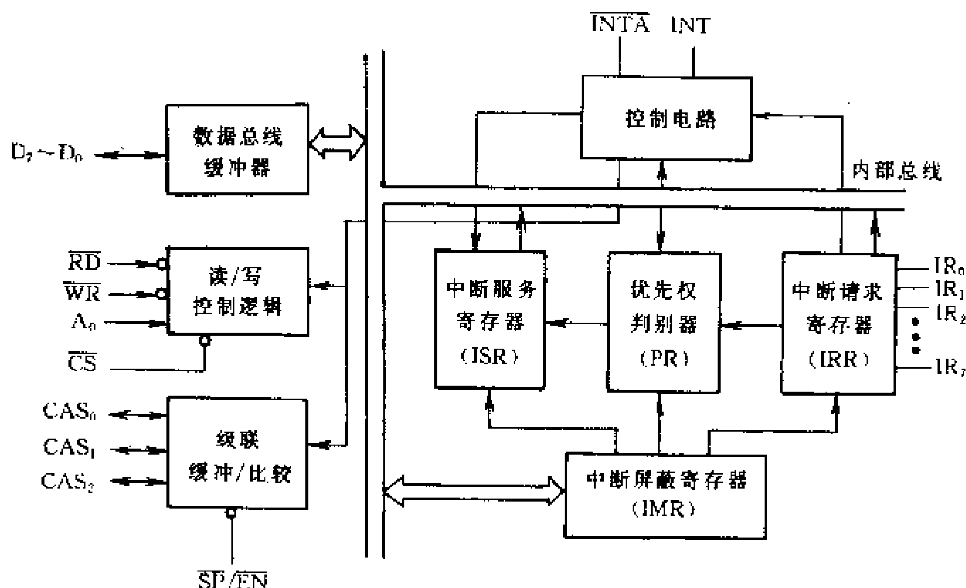


图 8-7　8259A 内部结构框图

(1)中断请求寄存器(IRR)

该寄存器用来存放由外部输入的中断请求信号 $IR_7 \sim IR_0$。当某个输入端为高电平时，该寄存器的相应位置"1"。

(2)中断服务寄存器(ISR)

该寄存器记录正在处理中的中断请求，当任何一级中断被响应，CPU 正在执行它的中断服务程序时，ISR 寄存器中的相应位置"1"，一直保持到该级中断处理过程结束为止。多重中断情况下，ISR 寄存器中可有多位被同时置"1"。

(3)优先级判别器(PR)

当输入端 $IR_7 \sim IR_0$ 中有多个中断请求信号同时产生时，由 PR 判定哪个中断请求具有最高优先级，并在 INTA 脉冲期间把它置入中断服务寄存器 ISR 的相应位。

(4)中断屏蔽寄存器(IMR)

该寄存器中存放有关被屏蔽中断的信息。当某位置为"1"时,表示禁止这一级中断请求进入系统,通过 IMR 寄存器可实现对各级中断的有选择的屏蔽。如图 8-8 所示。

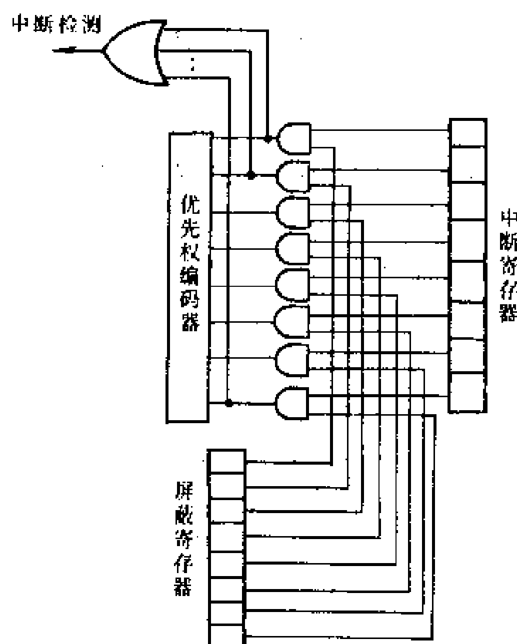


图 8-8 PIC 中断逻辑图

(5)级联缓冲/比较器

一片 8259A 只能接收 8 级中断,当超过 8 级时,可用多片 8259A 级联使用,构成主从关系。对于主 8259A,其级联信号 $CAS_2 \sim CAS_0$ 是输出信号,而对于从 8259A,级联信号 $CAS_2 \sim CAS_0$ 是输入信号。

此时,主 8259A 的 SP 端为"1",从 8259A 的 SP 端为"0",且从 8259A 的 INT 输出接到主的中断输入端 IR 上,因而最多可把中断扩展到 64 级。图 8-9 为三片 8259A 级联的连接图。

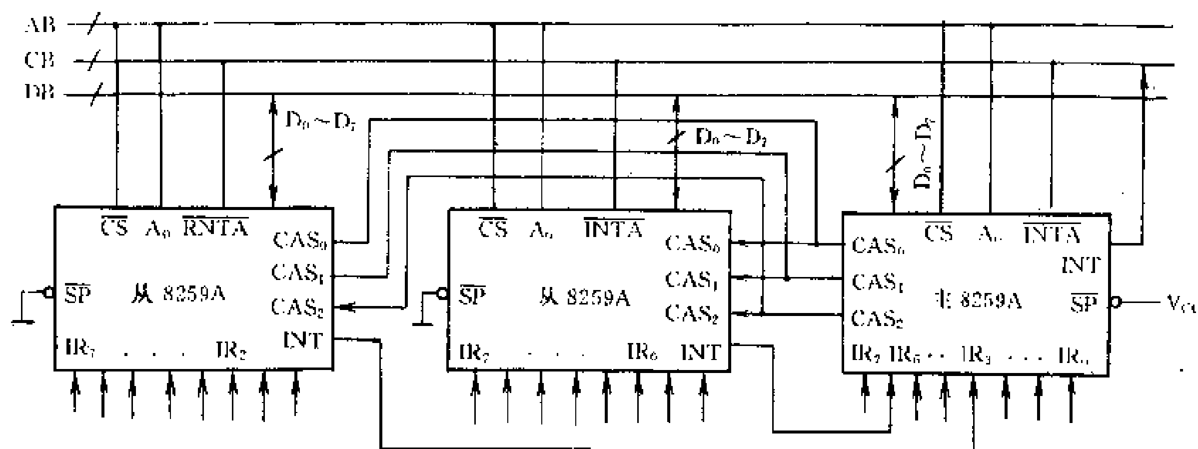


图 8-9 8259A 级连电路图

(6)控制电路

8259A 内部的控制电路,根据中断请求寄存器 IRR 的位置情况和优先级判别器 PR 的判

定结果，向 8259A 内部其它部件发出控制信号，并向 CPU 发出中断请求信号 INT 和接收来自 CPU 的中断响应信号$\overline{\text{INTA}}$，控制 8259A 进入中断服务状态。

(7)读/写控制逻辑

因一片 8259A 只占两个 I/O 端口地址，用地址线 A_0 来选择端口，端口地址的高位由片选信号端$\overline{\text{CS}}$输入。由读信号$\overline{\text{RD}}$和$\overline{\text{WR}}$信号控制可将命令写入有关的控制寄存器，或读出内部寄存器的内容。

(8)数据总线缓冲器

它是一个双向 8 位三态缓冲器，由它构成的 8259A 与 CPU 之间的数据接口。

2. 引脚分配

前面介绍了 8259A 芯片的内部结构，这些结构的功能由外部引脚的正确连接来实现。8295A 芯片的引脚可分为如下四类：

(1)与外部设备连接的中断请求输入引脚 $IR_0 \sim IR_7$；(2)与 CPU 连接的数据通路和控制信号：$D_0 \sim D_7$，$\overline{\text{WR}}$，$\overline{\text{RD}}$，$\overline{\text{INTA}}$，INT；(3)用于 8259A 级联的引脚 $CAS_0 \sim CAS_2$，$\overline{\text{SP}}/\overline{\text{EN}}$；(4)端口地址选择信号$\overline{\text{CS}}$，$A_0$。8259A 芯片的引脚分配如图 8-10。

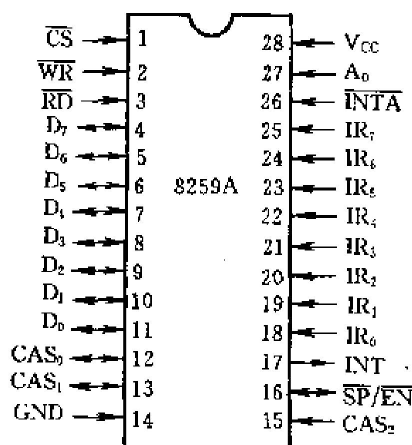


图 8-10　8259A 管脚图

二、8259A 的中断管理方式

8259A 具有非常灵活的中断管理方式，可满足使用者的各种不同要求。中断优先级的管理是中断管理的核心问题。8259A 对中断的管理可分为对优先级的管理和对中断结束的管理来讨论。

1. 中断优先级管理

8259A 对中断优先级的管理可分为四种情况：完全嵌套方式、自动循环方式、中断屏蔽方式和特殊完全嵌套方式。下面来讨论这四种方式。

(1) 完全嵌套方式

在此种方式下，8259A 的中断请求输入端引入的中断具有固定的优先级序列，IR_0 为最高优先级，IR_1 为次高优先级，依次类推，IR_7 为最低优先级。同时，高优先级的中断可中断低优先级的服务，实现嵌套中断。

(2) 自动循环方式

在完全嵌套方式下，中断请求 IR_0～IR_7 的优先级别是固定不变的，使得从 IR_0 引入的中断总是具有最高优先级，在某些情况下我们需要能改变这种优先级别。这时，可采用自动循环方式，在这种方式下，从 IR_0～IR_7 引入的中断轮流具为最高优先级。因为，当任何一级中断被处理完，它的优先级别就被修改为最低，而最高优先级分配给该中断的下一级中断。例如，现正为 IR_3 引入的中断服务，若服务完毕，IR_3 为最低优先级，IR_4 为最高优先级，IR_5 为次高优先级，依次排列。

(3) 中断屏蔽方式

用中断屏蔽方式管理优先级有两种方法：第一，普通屏蔽方式。这种方式是在中断屏蔽寄存器 IMR 中，将某一位或几位置"1"来屏蔽掉相应级别的中断请求，CPU 在执行主程序时将 IMR 寄存器的相应位置"1"，也可在 CPU 执行某级的中断服务中，禁止比它高的中断进入，在服务程序中将 IMR 寄存器的相应位置"1"屏蔽。第二，采用特殊屏蔽方式，这时可以使低优先级别的中断进入正在服务的高优先级别中。

(4) 特殊完全嵌套方式

特殊完全嵌套方式用在 8259A 有级联的情况。当任何一个 8259A 从片接收到一个中断请求，经本 8259A 判别确定为当前最高优先级，则响应这一中断，通过 INT 端向 8259A 主片相应的 IR 端提出中断请求。如果这时 8259A 主片中 ISR 相应位已置"1"，则说明该 8259A 从片的其它输入端已提出过申请，且正在服务。8259A 从片的判优电路判别到刚申请的中断优先级最高，故应停止现行中断服务去为刚申请的中断服务。8259A 有级联的情况下，按完全嵌套方式管理优先级。显然，接在主片 IR_3 上的从片比接在 IR_4 上的从片具有高的优先级，而主片上 IR_0、IR_1、IR_2 上的中断比接在主片 IR_3 上的从片具有高优先级。

2. 中断结束的管理方式

当 8259A 响应某一级中断而为其服务时，中断服务寄存器 ISR 的相应位置"1"，当有更高级的中断申请进入时，ISR 的相应位又要置" 1"，因而 ISR 寄存器中可有多位同时置"1"。在中断服务结束时，ISR 中相应位应清"0"，以便再次接收同级别的中断。中断结束的管理就是用不同的方式使 ISR 中相应位清"0"，并确定下面的优先排队。8259A 中断结束的管理分三种情况讨论：

(1) 完全嵌套方式：在中断嵌套过程中，中断服务寄存器 ISR 的内容在不断变化。例如：8259A 正在为 IR_7 的中断请求服务时，ISR 中的 D_7 位置" 1"，此时，有高级中断 IR_6 请求服务，若准许为其服务，ISR 中的 D_6 位又置"1"，假如又有更高级的中断，且为其服务，ISR 中将有对应位置"1"。总之，在完全嵌套方式下，最多可达 8 级中断嵌套，在中断服务结束时也应按从高到低的次序结束，每个中断结束，使 ISR 中相应位清"0"，待全部嵌套中断均结束，则 ISR 中每位均为"0"。

8259A 在完全嵌套方式下，可采用三种中断结束方式：

a. 一般 EOI 形式

当任何一级中断服务程序结束时，给 8259A 传送一个 EOI 命令，8259A 将 ISR 寄存器中级别最高的置"1"位清"0"。这种方式只有在当前结束的中断总是尚未处理完的级别最高的中断时，才能使用这种结束方式。如果在中断服务中修改过中断级别，则不能采用这种方式。

b. 特殊 EOI 方式

在一般 EOI 方式的基础上，当中断服务程序结束给 8259A 发出 EOI 命令的同时，将当前结束的中断级别也传送给 8259A，这就被称作特殊 EOI 方式。在这种情况下，8259A 将 ISR 寄

存器中指定级别的相应置"1"位清"0"。这种方式适合于任何情况下使用。

c. 自动 EOI 方式

CPU 进入中断响应总线周期的第二个中断响应信号$\overline{INTA}$结束时，自动将 ISR 寄存器相应置"1"位清"0"。中断结束时，不需要向 8259A 送 EOI 命令，这是一种最简单的结束方式。但是存在一个明显的缺点，任何一级中断在执行中断服务程序期间，在 8259A 中没有留下任何标志，如果在此过程中，出现了新的中断请求，则只要 CPU 允许中断，不管出现的中断级别如何，都将打断正在执行的中断服务而被优先执行，这显然是不合理的。因此，使用自动 EOI 结束方式，只有在一些以预定速率发生中断，且不会发生同级中断互相打断或低级中断打断高级中断的情况下。

(2) 自动循环情况

在这种情况下，也可采用三种中断结束方式，和前面完全嵌套的情况相同，分一般 EOI，特殊 EOI 和自动 EOI，但结束后的优先级处理有所不同。

a. 一般 EOI 方式

当任何一级中断服务处理完毕，给 8259A 送一个一般 EOI 结束命令，8259A 将 ISR 寄存器中级别最高的置"1"位清"0"。同时赋给它最低优先级，将最高优先级赋给原来比它低一级的中断请求，其它中断请求的优先级别以循环方式类推。

b. 特殊 EOI 方式

这种方式主要用在自动循环优先级管理方式下又有嵌套的情况。在一般 EOI 方式下，结束一个中断就将它置为最低优先级，若在服务程序中安排了一条优先级置位指令，使得优先级次序发生了变化，就必须使用特殊 EOI 方式，在向 8259A 发结束命令的同时，将其中断优先级别也传送给 8259A。这样，8259A 可根据用户的要求将最低优先级别给指定的中断源，其余中断源的优先级别按自动循环方式类推。

c. 自动 EOI 方式

在 CPU 响应中断的第二个$\overline{INTA}$结束时，自动将 ISR 寄存器中相应位清"0"，并立即按自动循环方式改变各级中断的优先级别。

(3) 特殊完全嵌套情况

这种情况是因为 8259A 有级联，因而 CPU 应发出两个中断结束命令 EOI，一个送主 8259A，用来将其主 8259A 的 ISR 寄存器相应位清"0"；另一个送从 8259A，用来将其从 8259A 中的 ISR 寄存器相应清"0"。

三、8259A 的编程

在使用 8259A 时，除按各引脚规定的信号接好电路外，还必须用程序选定其工作状态，例如各中断请求信号的优先级分配、中断屏蔽、中断矢量等等。每一种状态都由一个命令字或一个命令字中的某些位来规定。8259A 的命令字分为初始化命令字 ICW(Initialization Command Word)和工作命令字 OCW(Operation Command Word)两种，因此 8259A 的编程也分为初始化编程和工作编程两步。在 8259A 内部，有相应的一组寄存器分别将这些命令字锁存，以控制其工作。

1. 8259A 寄存器的读写

对于 8259A 内部的寄存器，除在编程序时 CPU 可用输出指令对它们逐一地写入外，在查询状态时还可用输入指令将其内容读出。为了寻址各个寄存器，除了用地址信号 A_0 译码外，

还需要用这些命令字的某些位作为访问某个寄存器的特征，或者按写入的先后顺序来进行区分。表 8-3 列出了对 8259A 各寄存器读写时的信号关系。

表 8-3　8259A 寄存器的读写

$\overline{CS}$	A_0	$\overline{RD}$	$\overline{WR}$	D_4	D_3	读 写 操 作
0	0	1	0	0	0	数据总线 → OCW_2
0	0	1	0	0	1	数据总线 → OCW_3
0	0	1	0	1	×	数据总线 → ICW1
0	1	1	0	×	×	数据总线→ ICW_2,ICW_3,ICW_4,OCW_1
0	0	0	1			IRR 或 ISR 或中断级别编码→数据总线
0	1	0	1			IMR → 数据总线

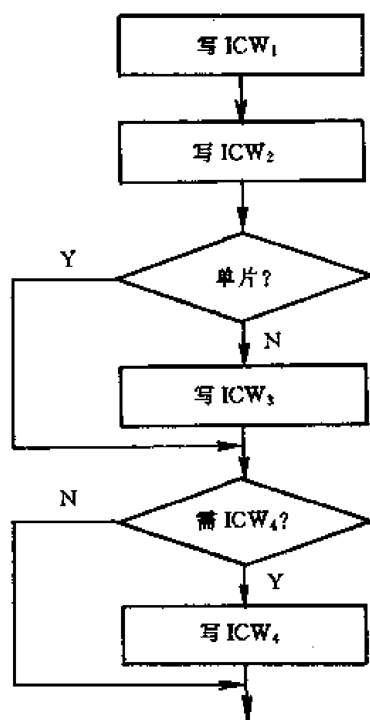


图 8-11　8259A 初始化顺序

表中上半部分用于编程写入，下半部分用于查询读出。其中 ICW_1～ICW_4 代表各初始化命令字寄存器，OCW_1～OCW_3 代表各工作命令字寄存器。在中断响应时 CPU 将从 8259A 读取中断矢量(由 ICW_2 事先写入 8259A 中)，这种情况是一种特殊的读出，表中没有列入。

2、8259A 的初始化编程

8259A 必须先进行初始化编程，后进行工作编程。初始化编程由写入 ICW_1 开始，然后写入 ICW_2。至于是否写入 ICW_3 和 ICW_4 取决于 ICW_1 和内容。其流程图如图 8-11 所示。

下面分析每个初始化命令字各位的作用。

(1)ICW_1

在地址 $A_0=0$ 时，若对 8259A 写入一个 $D_4=1$ 的字节，则启动了其初始化编程。写入的这个字节被当成 ICW_1，$D_4=1$ 是其特征，其余各位的作用见图 8-12。

D_7～D_5 和 D_2 这四位仅对 8080/8085 系统有意义。

$D_0=1$ 表示要送 ICW_4，否则不送。在不送 ICW_4 时，ICW_4 的各位默认值均为零。

$D_1=0$ 表示系统中有多片 8259A 级联，否则表示单片工作，此时不送 ICW_3。

D_3 规定 IR_7～IR_0 信号的触发方式。$D_3=1$ 为高电平触发，否则为上升沿触发。

写入 ICW_1 时，还自动将中断屏蔽寄存器 IMR 清零，并恢复各中断源的优先级为 IR_0 最高，IR_7 最低。

(2)ICW_2

ICW_2 是中断矢量基值寄存器，其作用示于图 8-13。在工作于 8086/8088 系统中时，D_7～D_3 表示中断矢量的高五位，D_2～D_0 不需编程，固定为 000，在响应时由中断源序号填入相应值。

(3)ICW_3

ICW_3 是 8259A 的级联命令字，单片 8259A 工作时不需写入。多片 8259A 级联时，有主片从片之分，需要分别写入 ICW_3。ICW_3 的作用如图 8-14 所示。主片 ICW_3 的 D_7～D_0 对应其八

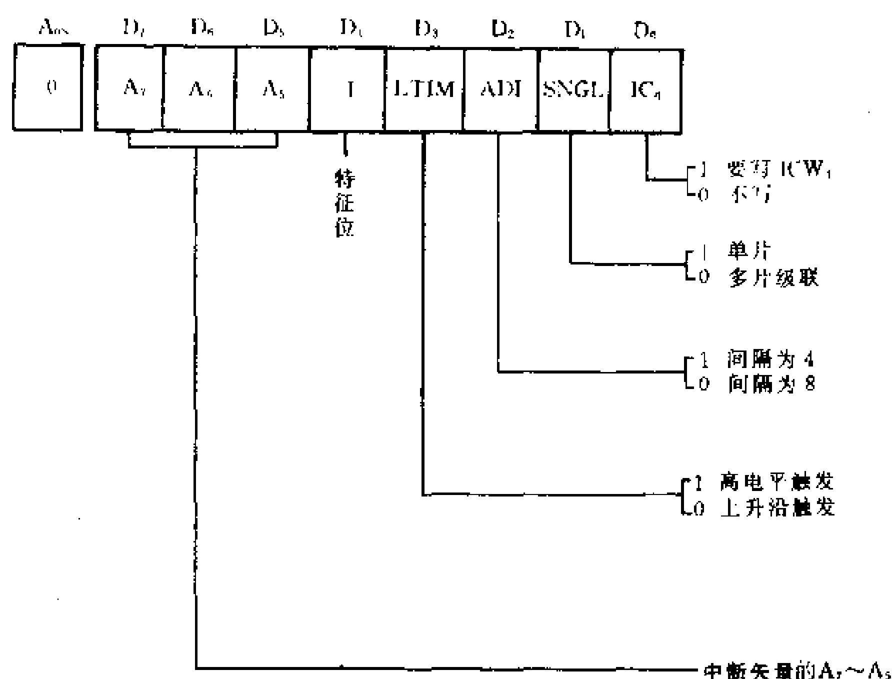


图 8-12　ICW₁ 的作用

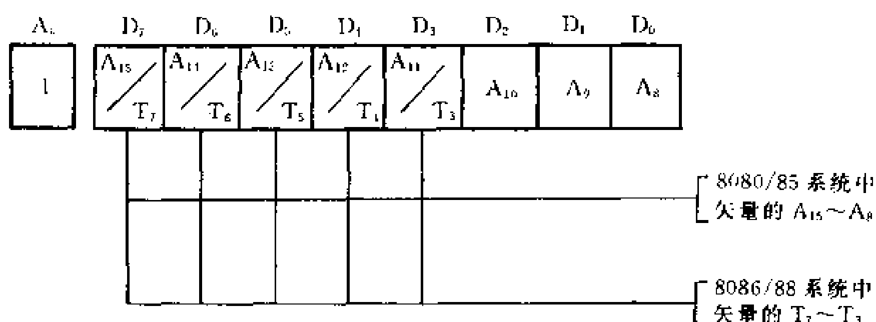


图 8-13　ICW₂ 的作用

条中断请求线 IR_7～IR_0，若某根 IR 线上接有从 8259A 片，则 ICW_3 的相应位应写成 1，否则写 0。各从片的 ICW_3 仅 D_2～D_0 有意义，作为其从片标识码，高位固定为 0，这个从片标识码须和本片所接主片 IR 线的序号一致。在中断响应时，主片通过级联线 CAS_2～CAS_0 送出被允许中断的从片标识码。各从片用自己的 ICW_3 和 CAS_2～CAS_0 比较，二者一致的从片被确定为当前中断源，才可发送自己的中断矢量。

(4)ICW_4

ICW_4 的作用见图 8-15，其高 3 位无意义。D_4 指定了中断的嵌套方式。$D_4=0$ 为一般嵌套方式，当某个中断正在服务时，本级中断及更低级的中断都被屏蔽，只有更高的中断才能响应。对于单片 8259A 的中断系统，这种安排没有问题。但对多片 8259A 级联组成的中断系统，当某从片中一个中断正在服务时，主片即将这个从片的所有中断屏蔽。因此即使本从片中有比正在服务的中断级别更高的中断源发出请求，也不能得到响应，即不能中断嵌套。$D_4=1$ 则是特殊嵌套方式，仅仅屏蔽比当前中断源低级的中断，于是上述情况就可以中断嵌套。

D_3 为数据缓冲选择。$D_3=1$ 时 8259A 的数据线和系统总线之间要加上三态缓冲器，此时

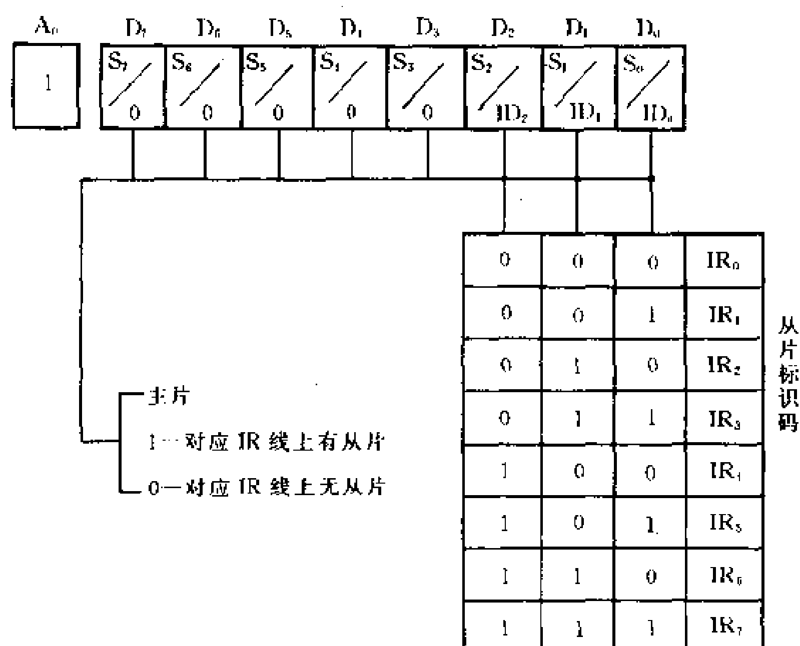


图 8-14　ICW_3 的作用

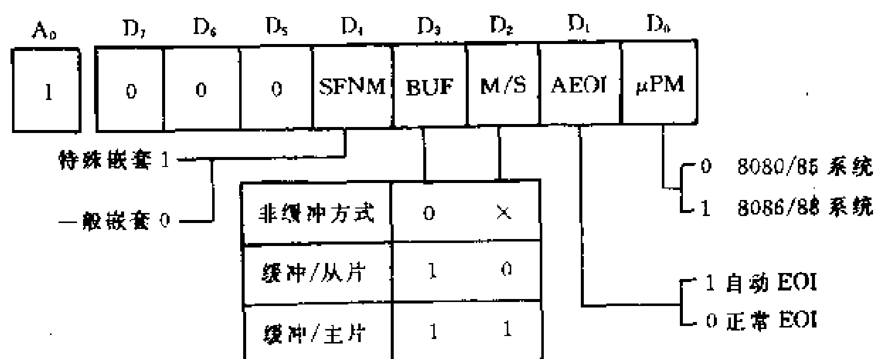


图 8-15　ICW_4 的作用

8259A 的$\overline{SP}/\overline{EN}$引脚变成输出线，以作为缓冲器的选通信号。每当 8259A 的数据送往系统总线时，$\overline{SP}/\overline{EN}$引脚输出有效的低电平。这种情况用于多片 8259A 的级联，此时主从片的区分就不能再靠$\overline{SP}/\overline{EN}$固定接高或低电平，而用 ICW_4 的 D_2 位。规定主片的 $D_2=1$，从片的 $D_2=0$。$D_3=0$ 时不加缓冲器，D_2 无意义。

D_1 说明了中断结束的方式。$D_1=0$ 是正常方式，即在中断服务结束时，向 8259A 送一个 EOI 命令字(OCW_2)，于是中断服务寄存器 ISR 中与中断源相对应的位被清除。$D_1=1$ 是自动 EOI 方式，即在中断响应时，在 8259A 送出中断矢量后，自动将 ISR 相应位复位。

D_0 指定了系统中所用 CPU 的模式，$D_0=0$ 时用 8080/8085CPU，$D_0=1$ 时用 8086/8088CPU。

若在某种应用场合，正好需要 ICW_4 各位都为 0，则可以不写 ICW_4。因为 8259A 在进入初始化时，已自动将 ICW_4 全部复位。四个初始化命令字必须按顺序写入，工作后一般不再重复写。

3. 8259A 的工作编程。

8259A 在初始化编程后，应再进行工作编程，即写入工作命令字。工作命令字有三个，它们各有自己的特征位，因此写入的顺序没有要求。在中断系统工作中，某些工作命令字可能需要重复多次地写入。

(1)OCW_1

OCW_1 又称中断屏蔽字，见图 8-16。三个工作命令字中仅 OCW_1 占有奇地址($A_0=1$)，其每一位控制一根中断请求输入线，屏蔽字为"1"的位，对应的中断请求线被屏蔽；否则，被允许。在初始化开始时，默认屏蔽字各位全为 0。

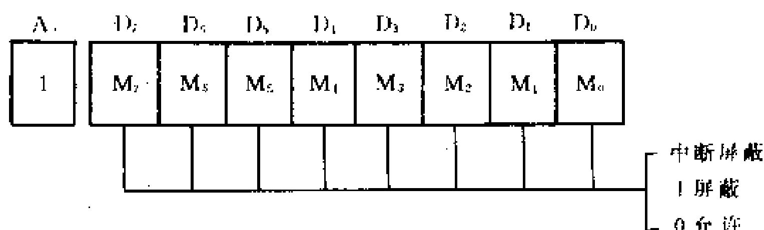


图 8-16 OCW_1 的作用

(2)OCW_2

写入 OCW_2 的主要作用是对 8259A 及中断结束命令，包括一般结束 EOI 和特殊结束 SEOI，其次还可以控制中断优先级的循环，见图8-17。它虽然和 OCW_3 都占有偶地址($A_0=0$)，但其特征为 $D_4D_3=00$，因此不会发生混淆。

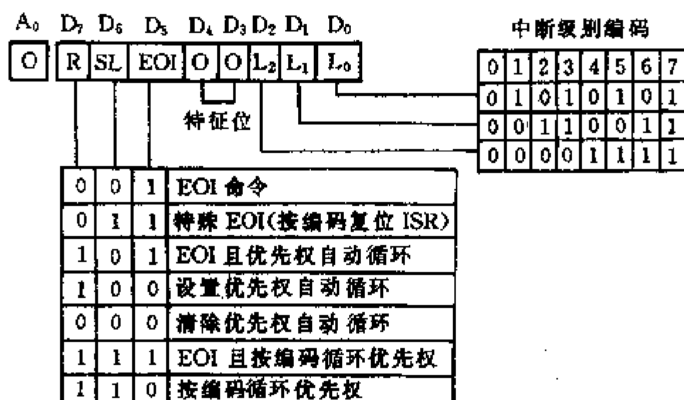


图 8-17 OCW_2 的作用

D_7 位表示中断优先级循环。当 $D_7=0$ 时，八个中断请求 $IR_7 \sim IR_0$ 的优先级固定不变，IR_7 最低，IR_0 最高。当 $D_7=1$ 时，优先级可以循环，即 IR_7 和 IR_0 首尾相接成一闭环，各级的优先级在其中循环移位。循环到什么情况停止，还与其他位有关。

D_6 位表示特殊循环。当 $D_6=1$ 时，最低三位 $D_2 \sim D_0$ 的二进制编码指定了一条外部中断请求线 IR_i($0<=i<=7$)。此时若 $D_7=1$，则称特殊循环，即优先级的循环移位一直进行到最低优先级对准 IR_i 为止，于是最高优先级也就移到 IR_{i-1}(若 $i=0$，则 $i-1=7$)。$D_6=0$ 时，最低优先级自动循环到当前服务的中断请求线，$D_2 \sim D_0$ 无意义。

D_5 位是中断结束位。$D_5=1$ 表示中断结束(EOI 命令)。当用 8259A 来实现中断管理时，中断服务程序结束时(返回指令 IRET 前)，必须给 8259A 送一条 EOI 命令(即 $D_5=1$ 的 OCW_2)。8259A 收到这条命令后，将中断服务寄存器 ISR 中的相应位清除，然后才为其他中断源服务。若 $D_6D_5=11$，则称为特殊的中断结束(SEOI 命令)，它将复位 ISR 中由 OCW_2 的 $D_2 \sim D_0$ 编码指定的位。

(3) OCW_3

写入 OCW_3 的地址和 OCW_2 相同，但其特征为 $D_4D_3=01$。其各位的作用见图 8-18。

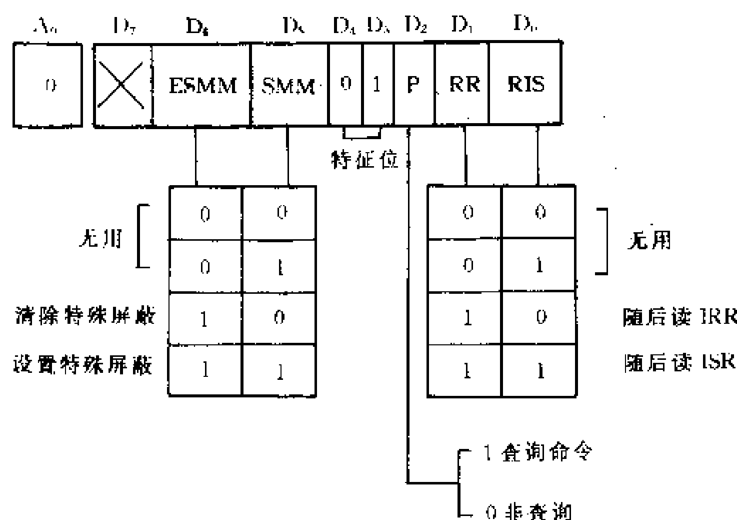


图 8-18 OCW_3的作用

OCW_3 经常用来配合读 8259A 内部寄存器的内容。在对 8259A 写入 OCW_3 的 $D_1D_0=10$ 之后，对同一个地址($A_0=0$)再作输入指令，则将读入其中断请求寄存器 IRR 的内容；若在写入 $D_1D_0=11$ 的 OCW_3 后，也对同一个地址($A_0=0$)作输入，则读入的是中断服务寄存器 ISR 的状态。

除以上两个寄存器以外，任何时候对 8259A 用奇地址($A_0=1$)作输入，都可以读入其中断屏蔽寄存器 IMR 的内容，而不必先送 OCW_3。

OCW_3 中的 D_2 位表示查询。8259A 也可以不工作于中断方式，而工作于查询方式。此时应写入 $D_2=1$ 的 OCW_3，然后 CPU 再对同一个地址($A_0=0$)作输入时，就得到该片 8259A 的一个状态字节如下：

其中若 $D_7=1$，表示本片 8259A 的 IR_7～IR_0 中发生了有效的请求，此时优先级最高的外部请求的编码由 D_2～D_0 给出，若 $D_7=0$，则本片没有外部有效的请求发生。

CPU 可以反复对 8259A 查询，但每次查询前都应先送一次 $D_2=1$ 的 OCW_3。

OCW_3 的 D_6D_5 用来控制特殊屏蔽功能。当 $D_6D_5=11$ 时，设置特殊屏蔽；当 $D_6D_5=10$ 时，清除特殊屏蔽。当一个优先级较高的中断源正在服务的过程中，若设置了特殊屏蔽功能，则允许优先级较低的中断源产生中断嵌套。

4. 8259A 的应用

(1)在某个 8086 最小方式系统中接有一片 8259A，有一外设中断请求从 IR_7 引入，8259A 的端口地址及外设中断类型号由图 8-19 给出。

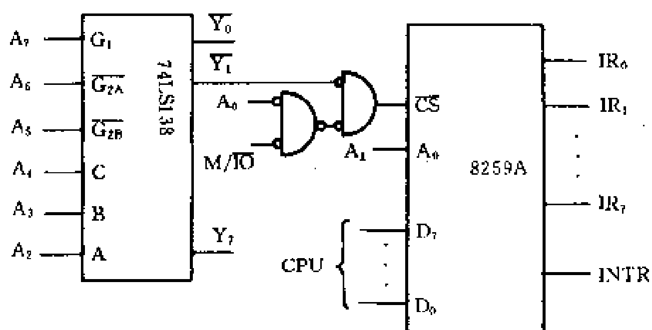


图 8-19 8259A 端口地址形成电路

从图 8-19 可知，8259A 具有两个端口地址，由 CPU 的地址线 A_1 控制，其端口地址的高位由译码器的输入端$\overline{Y_1}$提供，组合逻辑电路可使其

8259A 为偶地址，因而 8259A 的命令和类型号等可由 8086CPU 的低 8 位数据线传送。设中断为边沿触发，从 IR_7 引入中断的中断类型号为 C7H，端口地址为 84H 和 86H，8086CPU 获得中断类型号并乘 4，到中断服务程序入口地址表中找到相应的中断服务程序入口地址，而后转入中断服务。初始化程序包括对 8259A 的预置，以及中断服务程序首地址填入中断矢量表中。

```
INTRRUP SEGMNET AT 0
        ORG   0C7H*4
INTC7   LABEL  DWORD
        ⋮
MAIN    SEGMENT
        ⋮
        CL1                   ;关中断
        MOV   AL,13H          ;写 ICW1
        OUT   84H,AL
        MOV   AL,0C7H         ;写 ICW2
        OUT   86H,AL
        MOV   AL,01           ;写 ICW4
        OUT   86H,AL
        STI                   ;开中断
        ⋮
MAIN    ENDS
        ⋮
```

(2)PC/XT 启动时执行 BIOS 中系统初始化程序，其中包括 8259A 的编程(初始化编程和工作编程)。

有关部分程序如下：

```
        ⋮
        MOV   AL,13H          ;写 ICW1
        OUT   20H,AL
        MOV   AL,8            ;写 ICW2
        OUT   21H,AL
        MOV   AL,9            ;写 ICW4
        OUT   21H,AL
        MOV   AL,0FFH         ;写 OCW1
        OUT   21H,AL
        ⋮
```

程序中对 8259A 写入的 ICW_1＝13H＝00010011B，表明外部中断请求信号为上升沿有效，单片 8259A 工作，且后面还要写 ICW_4。写入的 ICW_2 是中断矢量，现为 8，实际上 8 个中断源各自填入 D_2～D_0 三位，形成 08～0FH 八个矢量。ICW_4＝ 9＝ 00001001B ，指定了系统中的 CPU 为 8086/8088，中断过程不自动结束，应写入一个含 EOI 命令的 OCW_2 结束，数据线上有

缓冲器，且取一般中断嵌套方式。最后送入的 OCW_1 = 0FFH，屏蔽所有硬中断，因为系统尚未初始化完毕，不能接收任何中断。

四、8259A 与 PC 机的硬件中断

在 PC 机中，CPU 的一个基本任务是响应中断。例如，系统时钟，键盘以及磁盘控制器等所有的硬件在不同的时间均产生中断。由于每个硬件中断在 ROM BIOS 或在 DOS 中具有相应的中断处理程序，所以 CPU 可以识别和响应每个中断。

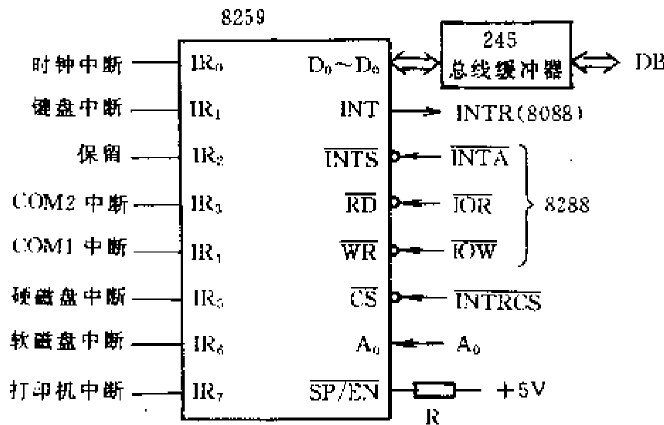


图 8-20　8259A 的连接

8259A 在 IBM PC/XT 中的连接如图 8-20 所示。数据线 $D_7 \sim D_0$ 经过 74LS245 总线缓冲器后接系统数据总线。中断请求信号 INT 直接连 CPU 的 INTR 端。由于 CPU 是处于最大模式下，其中断响应信号 $\overline{INTA}$ 和 I/O 端口的读写信号 $\overline{IOR}$，$\overline{IOW}$ 由总线控制器 8288 产生，所以 8259A 的这三端都和 8288 对应端相接。8259A 的片选信号 $\overline{CS}$ 接 I/O 口的译码输出 $\overline{INTRCS}$，占用地址 20H 和 21H。其 A_0 接地址总线 A_0。由于在 IBM PC/XT　BIOS 自举时，对 8259A 编程为数据输出取缓冲方式，故 8259A 的 $\overline{SP}/\overline{EN}$ 端成为输出信号，在系统中未使用，仅用电阻 R 接高电位。

系统将 8259A 的八条中断输入线占用了七条，所接外部电路的名称见图 8-20。其中时钟中断的级别最高，键盘次之，而并行打印机中断的级别最低。IR_2 保留未用，并已引至系统总线 B_4 端，即 IRQ_2 信号，可供用户扩展硬件中断功能使用。

根据片选信号和地址 A_0 的译码，8259A 内部各寄存器的读写地址如下所列：

写入操作

20H：ICW_1、OCW_2、OCW_3

21H：ICW_2、ICW_3、ICW_4、OCW_1

读出操作

20H：中断请求寄存器 IRR、中断服务寄存器 ISR、中断级编码（查询方式时）

21H：中断屏蔽寄存器 IMR

在 PC/XT 机中，有一片 8259A 中断控制芯片，可管理 8 级中断，I/O 端口地址为 020H 、021H，从 $IR_0 \sim IR_7$ 分配的中断类型号为 08H～0FH。在 PC/AT 机中有两片 8259A 中断控制芯片，主片的端口地址和中断类型号的分配与 PC/XT 机相同。增加的 8259A 为从片，端口地址为 0A0H 和 0A1H，从 $IR_8 \sim IR_{15}$（从片的 8 根中断引线）分配的中断类型号为 70H～77H。

PC 机中的最高优先级中断（IR_0）为定时节拍中断，定时器以每秒产生 18.2 次的速率向 CPU 发出中断请求。其中断服务包含两项内容：一是计时；二是管理软磁盘驱动器的启用时间。

PC 机中次高级优先级中断（IR_1）为键盘中断。键盘接口接收从键盘发来的串行数据，在移位寄存器中获得按键的扫描码。串行数据的最后一位进入移位寄存器后，就向 CPU 发出中

断请求。键盘在每一次按键过程中发送两个码：按下时发送的是“通码”；放开弹回时发送的是“断码”。断码的特征是最高位为1。键盘中断处理程序从接口中读入键盘扫描码，除SHIFT、CTRL、ALT几个键外，仅对“通码”进行处理，“断码”不处理。普通字符键的扫描码转换成相应的ASCII码，存入键盘输入缓冲区。一些不能用ASCII表示的键，如→、←、↑、↓等键，则以扩展码的形式存入键盘输入缓冲区。其它控制键则存入键盘状态单元。

主片上的IR_2接有从片8259A。在PC/AT机中，中断扩展到15级，而PC/XT中仅有8级，PC机硬件中断的引入及中断源如表8-4所示。

表8-4 PC机中断源及引入表

中断引入引脚		中断源
主片	IR_0	计数器0通道输出
	IR_1	键盘中断
	IR_2	从片8259A
	IR_3	网络通讯
	IR_4	串行通讯
	IR_5	硬盘控制
	IR_6	软盘控制
	IR_7	打印机控制
从片	IR_0	定时时钟
	IR_1	IRQ_9引入
	IR_2	IRQ_{10}引入
	IR_3	IRQ_{11}引入
	IR_4	IRQ_{12}引入
	IR_5	IRQ_{13}引入
	IR_6	IRQ_{14}引入
	IR_7	IRQ_{15}引入

第四节 8086中断矢量表的建立

前面我们讲到8086利用矢量中断的方法，一旦响应中断可方便地找到中断服务程序的入口地址。它是在规定的内存区中，每四个连续字节存放一个中断服务程序首地址，可建立一个1K字节大小的表。尽管表规定了内存区域，但表中的内容，即中断服务程序入口地址是用户任选的。为了让CPU响应中断后正确转入中断服务，中断矢量表的建立是非常重要的。我们介绍四种建立这个表格的方法。

一、绝对地址置入法

我们知道指示性语句AT和ORG均可指定存储单元的绝对地址。AT可指定段基值(16

位)，但不能指定代码段，而ORG将指定偏移地址。中断矢量表的段基址，可以用下面指令设定 INT-TBL SEGMENT AT 0，中断类型号乘4为该段的偏移地址，可用ORG n*4设定。然后可用DD伪指令将中断服务程序的首地址装入。

```
INT—TBL SEGMENT AT 0
        ORG n*4
        DD  INT  VCE
 INT—TBL ENDS
           ⋮
MCODE SEGMENT           ;主程序
           ⋮
INT—VCE PROC FAR        ;中断服务程序
           ⋮
    IRET
           ⋮
```

二、使用串送存指令装入法

使用串指令STOS该指令是将AX/AL寄存器内容写入附加段由DI所指向的目标偏移地址单元中。我们只要将ES设定为0，DI中设定为n*4的内容，使用STOSW指令，即可完成中断服务程序首地址的装入。

```
    ⋮
CLI                             ;禁止中断
MOV   AX,0
MOV   ES,0
MOV   DI,n*4H                   ;置矢量表段地址
MOV   AX,OFFSET INT_VCE         ;置矢量表偏移地址
CLD
STOSW
MOV   AX,SEG INT_VCE            ;中断服务程序段地址
STOSW
STI
    ⋮
```

三、使用DOS调用

利用DOS中断21H以及专门为更新中断服务程序入口地址的25H号功能来设置中断地址有两个非常显著的优点：其一是DOS会采取措施用最安全可行的方法来存放中断矢量；其二是使用时范围更广泛。

使用25H功能时要求：AL＝中断类型号，DS:DX＝中断服务程序入口地址的段，偏移地址。下面程序完成中断类型号为60H的中断地址置入。

```
PUSH    DS                  ;保存当前数据段
```

```
MOV    DX,SEG INT60H
MOV    DS,DX
MOV    DWX,OFFSET INT60H
MOV    AL,60H
MOV    AH,25H
INT    21H
POP    DS
```

由于系统的中断类型号 0～255 中，有许多类型号已由系统使用。如果用户企图修改某一中断服务程序的首地址，应将原有的中断服务程序入口地址用 DOS 调用的功能 35H 保存起来，以便从用户程序中退出来时，再用 25H 功能恢复，而不影响系统的正常工作。

DOS 调用 21H 中断的 35H 号功能是：对指定的机器中断，得到当前中断处理程序的入口地址。AL＝中断类型号，返回时，ES 中有段基址，而 BX 中有偏移地址，下面例子是对机器的中断类型 0 号，将其当前中断服务程序入口地址取出，并保存到变量 INTOSEG 和 INTOFF 中。

```
MOV   AH,35H             ;功能号置 AH 中
MOV   AL,0               ;中断类型号置 AL 中
INT   21H
MOV   INTOSEG,ES         ;保存原中断服务程序段基址
MOV   INTOFF,BX          ;保存原中断服务程序偏移地址
        .
        .
        .                ;可用 25H 重新设中断服务程序地址
        .
        .
        .
INTOSEG DW ?
INTOFF  DW ?
```

下面是用户更改系统中断类型 1CH 而写入新的中断服务程序的例子：

```
DATA   SEGMENT
CC1   DW 0
SS1   DW 0
DATA   ENDS
CODE   SEGMENT
MAIN PROC FAR
      ASSUME CS:CODE,DS:DATA,ES:DATA
START:  PUSH   DS
        SUB   AX,AX
        PUSH   AX
        MOV   AX,DATA
```

```
        MOV   DS,AX
        MOV   AL,1CH              ;保存旧的中断地址
        MOV   AH,35H
        INT   21H
     PUSH  ES                     ;旧中断地址入栈
        PUSH  BX
        PUSH  DX
        MOV   DX,OFFSET CLINT     ;建立新中断地址
        MOV   AX,SEG CLINT
        MOV   DS,AX
        MOV   AL,1CH
        MOV   AH,25H
        INT   21H
        POP   DS
          ⋮
MAIN    ENDP
CLINT   PROC NEAR                 ;新的中断服务程序
        PUSH  DS
        PUSH  BX
          ⋮
CLINT   ENDP
CODE    ENDS
  END   START
```

四、直接装入法

若外设的中断类型号为 6BH，则此中断类型号对应的中断矢量表地址为从 001ACH 开始的四个存储单元。设中断服务程序段地址在 1000H，偏移地址为 2000H。我们可用传送指令将已知的中断服务程序首地址置入中断矢量表中

```
    ⋮
XOR   AX,AX
MOV   DS,AX                     ;指向 0 段
MOV   AX,2000H
MOV   WORD PTR[01ACH],AX        ;置偏移地址
MOV   AX,1000H
MOV   WORD PTR[01ACH+2],AX      ;置段基址
    ⋮
```

第五节 82380中断控制器介绍

82380 芯片上集成有三个比 8259A 功能强的可编程中断控制器,简称 PIC。图 8-21 所示即为 82380PIC 的结构框图。由图看出三个控制器共有 15 个外部和 5 个内部中断请求线。每一个外部中断请求输入线又可以与另一个 8259A 从控制器串接,因此使得 82380PIC 最多可具有 15×8=120 个外部中断请求输入。通常 PIC 的三个控制器均可操作于主形态。且优先顺序为 A、B 和 C,即 A 控制器优先级最高,C 控制器优先级最低。

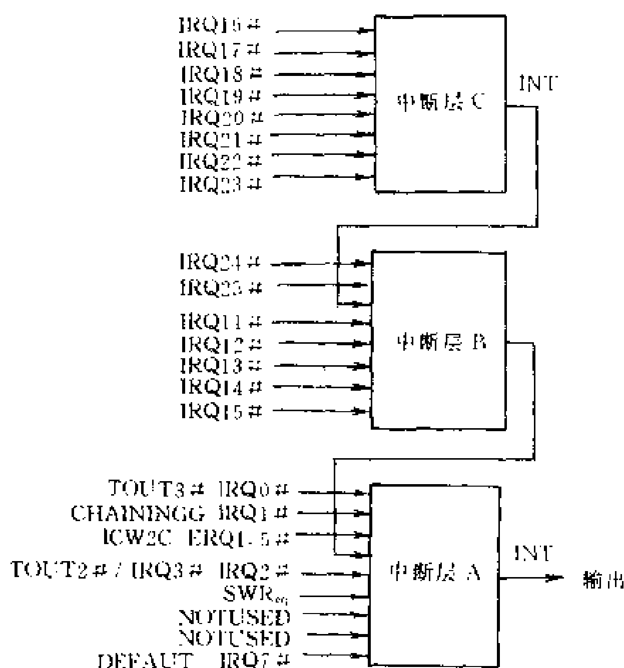


图 8-21 82380 PIC 结构

在有一个或一个以上的中断请求发出时,82380PIC 即会自动选择目前在等待的所有中断请求中优先顺序最高者,将其中断向量经数据总线送给主 CPU80386/80486,由主 CPU 为该中断请求服务。

与 8259A 相比,82380 PIC 最大的改进之处就是每一个中断请求输入,均可独立设计其中断向量。在中断向量映射上更有灵活性,当有错误的中断响应周期发生时,PIC 会自动送出一个既定的向量,该向量可以由系统软件加以设计,它可以让主 CPU 知道,目前并无外设提出中断请求。

除此之外,82380 PIC 的功能与 8259A 完全相同。

一、PIC 功能与内部结构

在内部结构上,由图 8-21 可知,82380PIC 共含有三个相当于 8259A 的中断控制器:A,B,C,而且彼此串接。对外中断请求线由 A 中断控制器向主 CPU 发出。中断控制器 A 有 9 个中断请求输入(其中两个未用),中断控制器 B 和 C 各有 8 个中断请求输入。在 15 个中断请求输入中,有两个是其他功能共享的,其中 IRQ3# 可用作第二计数器的输出 TOUT2#,此一引脚有三种不同的用法:只用于 IRQ3# 的输入;只用于 TOUT2# 输出和以 TOUT2# 产生 IRQ3# 中断请求。此外 IRQ9# 可用于 4 号 DMA 请求输入(DREQ4)。通常 IRQ9# 与 DREQ4# 在某一瞬间只能使用其中一个。

图 8-21 所示的 82380PIC 中断控制器的 5 个内部中断请求为:

IRQ0#: 连接第 3 计数器输出(TOUT3#)
IRQ8#: 连接第 0 计数器输出(TOUT0#)
IRQ1#: 连接 82380DMA 连锁请求
IRQ4#: 连接 82380DMA 终止计数请求
IRQ1,5#: ICW_2 写入中断请求

另外,所有中断请求输入 IRQ 均为低电平有效,而且可以通过设置第一控制字(ICW_1)的一个控制初始化编程为边沿触发或电平触发。15 个中断请求输入引脚均有内部提升电阻。图 8-22 为每一个中断控制器内结构框图。由图看出,除可独立设计中断向量寄存器外,其结构完

全与 8259A 相同。即中断请求寄存器 IRR 存放中断请求输入(IRQ) 线上的中断请求。中断服务寄存器 ISR 存放目前正在被服务的中断请求。优先级判别电路 PR 是用来比较和判断 IRR 中哪一个中断请求优先级最高并选择此优先级最高位，在中断响应周期时，将其送入 ISR 中的对应位内。使该 IRR 位所对应的中断请求被服务。中断屏蔽寄存器是用于寄存哪些中断请求输入线应被屏蔽掉(即禁止中断)。其中主要是针对 IRR 屏蔽掉一个优先级高的中断请求输入，并不影响优先级低的中断请求输入的服务。中断向量寄存器含有 8 个寄存器(8 位)，每一个中断请求线对应一个，该寄存器用以存放预先设计好的中断向量(中断类型码)。此一中断向量在中断响应周期时被置于数据总线上。而控制电路则用来控制每一中断部件的操作。在有一个或一个以上未被屏蔽掉的中断请求输入操作时，该电路即会令中断输入信号 INT 高电平有效。此信号经串接，然后由中断控制器 A 的 INT 加至主 CPU80386/80486。此控制电路同时可以接收中断响应周期信号，并令对应的中断向量寄存器工作。A 中断控制器也负责处理特殊的 ICW_2 中断请求输入(IRQ1,5)。

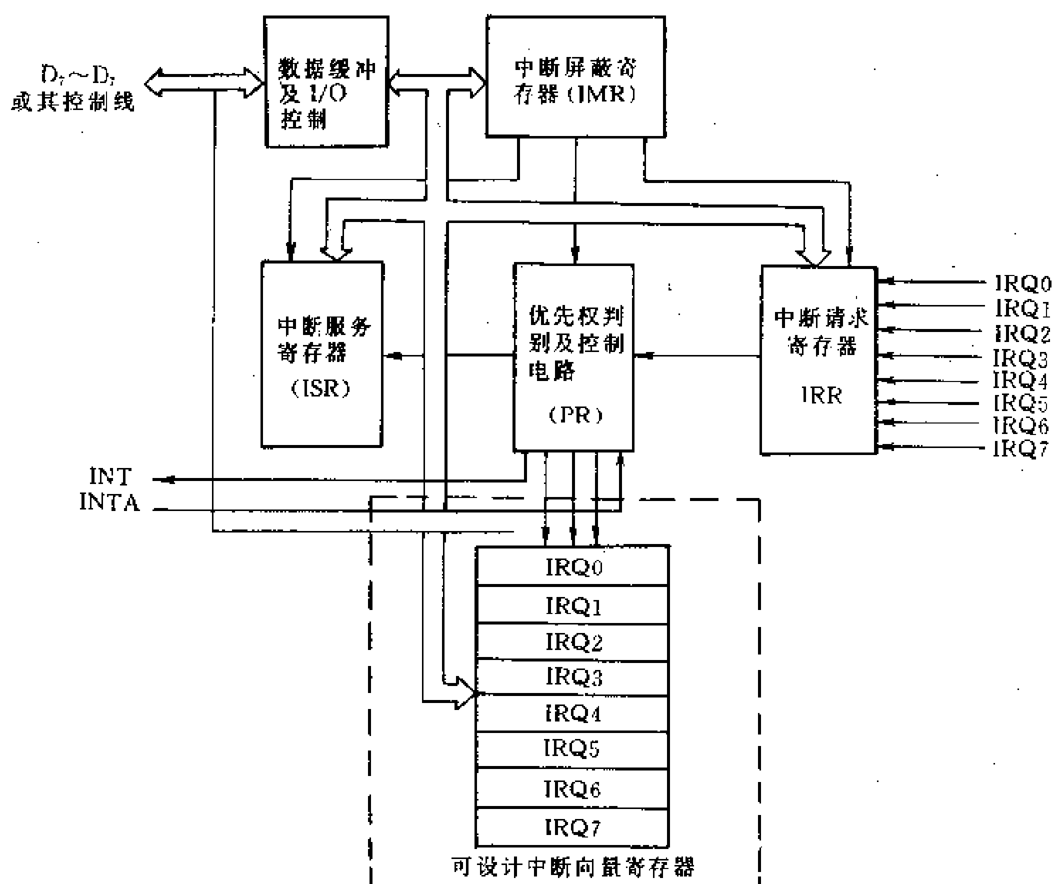


图 8-22 82380 PIC 中断控制器结构框图

二、82380 PIC 中断响应时序

当 82380PIC 的中断请求输出 INT 高电平有效后，若当时主 CPU 80386/80486 的 INTR 中断被响应，则主 CPU 会执行两个连在一起的中断响应周期，以响应 PIC 的中断请求。图8-23 所示即为中断响应时序图。

在使 INT 输出高电平有效后，82380 一直监视总线状态信号(M/IO#，D/C# 和 W/R

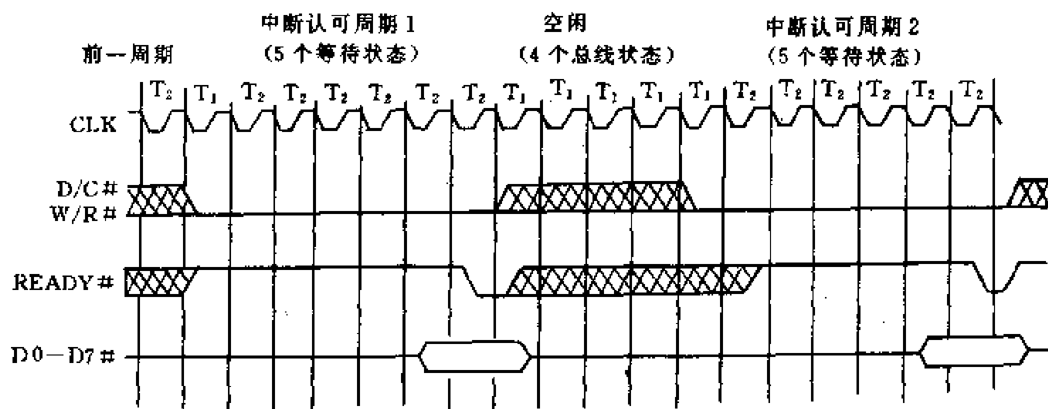


图 8-23　82380PIC 中断响应时序图

#),并等待主 CPU 开始第一个中断响应周期。即在第一个$\overline{INTA}$周期时,82380PIC 会选出一个最高优先的中断请求。假若此中断请求输入未串接任何从中断控制器,则在第一个$\overline{INTA}$周期时,82380 即会将 00H 置于数据总线上。在第二个$\overline{INTA}$周期时 PIC 则会将相应的中断向量值置于数据总线上。

若 82380PIC 由控制字寄存器 ICW_3 得知目前的中断请求输入串接一外部中断请求从控制器,则在第一$\overline{INTA}$周期时,PIC 即会将该串接部件的地址(而不是 00H)置于数据总线上。此一从串接地址已事先存入中断向量寄存器内,即为中断向量。而在第二个$\overline{INTA}$周期时,数据总线则处于浮空状态,这样就保证了从中断控制器的中断向量能够送至数据总线上,得以响应。

为了得到主 CPU 正确的响应和服务,中断请求输入 IRQ 的有效电低平必须一直保持至第一个$\overline{INTA}$周期开始为止。若在第一个$\overline{INTA}$周期来临时,届时已无中断请求,则 82380PIC 即会产生一既定向量,此即为中断控制器 A 的 IRQ7 的向量。

根据主 CPU80386/80486 的总线周期定义,两个$\overline{INTA}$周期之间会有四个总线空闲状态。主 CPU 会自动加入这些空闲状态。此外,在每一个中断$\overline{INTA}$周期,82380 内部等待状态产生其内部延迟所必须的等待状态。

三、82380 PIC 的操作形态

82380PIC 有许多种操作形态和控制命令。这些都是可编程的,即可通过程序随时更改或设置。即每一个中断请求输入都可以操作于用户所选择的形态。

1. 结束形态

在 80386/80486 主 CPU 执行完一中断服务程序后,82380PIC 必须得知此状态,以便更新中断服务寄存器(ISR)的内容。这样,82380PIC 才能随时得知目前那一种中断层次被服务,对应的优先顺序如何。中断结束形态简称 EOI,有三种不同的模式:不明确的 EOI 模式,明确的 EOI 模式和自动的 EOI 模式。选择使用哪一种 EOI 视用户所要执行中断操作的不同而定。

若 82380PIC 不是设置成自动 EOI 模式,则主 CPU 必须送出一个 EOI 命令,给一个明确的 82380 中断控制层,同时此中断层串接到另一内部层,则此内部层也应收到相同的 EOI 命令。举例而言,若 80386 目前刚服务完的是中断层 C 中的一个中断请求,则 EOI 命令必须同时

送给(即写入)A,B与C层。若C层的中断请求进一步来自另一串接到C层的外部从中断控制器(如8259A)。则除了A,B和C层外,该外部中断控制器(8259A)也应写入这一EOI命令。

不明确EOI模式。主CPU所送出的不明确EOI命令是首先让PIC中断层知道有一个中断被处理完,但不指明是那一个中断层次,有关的中断层必须自己决定中断层次,且清除对应的中断服务寄存器ISR位。

为了采用不明确EOI模式,有关的中断层必须操作于能事先得知正在接受服务处理的程序的层次的形态。因此不明确EOI命令只能用在刚服务且层次恒为最高优先层次时(也即,全嵌套形态结构时)。这样在中断层收到不明确EOI命令时,即可亲自将最高优先层次的ISR位清除为0,表示最高优先的中断处理程序已被执行完。因此在使用不明确EOI命令时,必须考虑清楚。由于全嵌套形态结构会被破坏掉,因此,在下面两种情况下,最好不用不明确EOI命令,即:(1)中断处理程序内有使用设置优先顺序的命令;(2)使用特殊的屏蔽形态。

明确EOI模式。明确EOI命令,每次均明确地指出那一个ISR位应被清除为0,它与不明确模式每次恒清除最高优先的ISR位不同。此模式的命令字可以指明一中断层的任一IRQ层次,在中断层次无法自动得知中断处理程序被执行完时应清除哪一ISR位时,明确EOI命令可派上用场。该命令可用于任何状况下。

自动EOI模式。在PIC设置成自动EOI模式时,主CPU80386/80486不再需要送出EOI命令通知中断层已执行完一个中断处理程序。中断层会在第二$\overline{INTA}$周期自动执行一不明确的EOI命令,以完成此操作。使用这种形态必须注意的是很可能会干扰到全嵌套形态结构。在自动EOI模式时,被执行的中断处理程序ISR位,在中断被响应之后,立即被清除为0,因此,ISR内不再记载那一个处理程序正被执行,此时若在同一中断层内正好有中断请求发生,而且主CPU中断允许,则不论其优先顺序如何,马上会执行。因此使用这一模式时,在执行中断处理程序之际,主CPU应将其中断请求输入禁止。这样,更高优先的中断层次只有在目前的中断处理程序执行完成时,才会被处理。全嵌套形态的结构才可确保,正被执行的中断处理程序即可一路执行完毕,而不会中间被中断。

2. 中断优先顺序

82380PIC提供各种中断请求输入方法,以适应各种不同的应用。这些方法是:全嵌套形态,自动循环形态和明确循环/明确优先顺序。

全嵌套形态操作是一种通用的优先顺序安排方法。这种形态提供一种多层的中断结构,中断层中的所有中断请求输入的优先顺序为:由最高排到最低,在初始编程后,82380PIC自动进入既定的全嵌套形态。此时,优先顺序是IRQ0#最高,依次排到IRQ7#最低。

每次在主CPU响应中断时,82380PIC会自动由中断请求寄存器(IRR)中找出最高优先的中断请求,并将其中断向量送到数据总线上。同时,PIC也会将中断服务寄存器(ISR)中对应位的值置1。以显示该中断请求正被服务。此ISR位值为1会一直保持到主CPU80386/80486执行完该中断处理程序要返回而送出EOI命令时。不过,若处于自动EOI模式(即自动EOI中断AEOI位值为1),则ISR位值将在第二$\overline{INTA}$周期被清除为0。

自动循环式优先顺序用在中断层内的所有设备的中断优先顺序都均等。使用这种方法时,某一设备一旦被服务,其优先顺序即指定为最低,使其他的设备都有机会先被服务。

自动循环式优先顺序有两种方法完成:一是与不明确EOI命令合用,另一种是与自动EOI模式合用,与不明确EOI指令合用时,自动循环式优先顺序控制则以一称为不明确EOI循环命令完成 。在此命令下达时,除了最高优先的ISR位会像在正常的不明确EOI命令时被

清除为 0 外，此 ISR 位所对应的中断层次同时被设置成最低优先。其他 IRQ 的优先顺序则顺着循环，一直转至最低优先顺序对象刚被服务完为止。循环定位以后定义的优先顺序，即为各 IRQ 的新优先顺序 。

与自动 EOI 模式合用时，在自动 EOI 循环的情况下，优先顺序循环变换早在中断请求的第二中断响应周期时即已发生。要进入或要离开此操作形态可以分别利用自动 EOI 循环设置命令与自动 EOI 循环清除命令。一旦进入自动循环式形态，则不正常的 EOI 形态还需要其他的命令 。要提醒的是使用这种自动循环模式需特别小心。

明确循环/明确优先顺序方法主要应用于已知某一外设的中断优先顺序必须更改时，与自动循环式不同的是，明确循环式完全由使用者控制。使用者可以选择哪一中断层次为最低或最高顺序。可在主程序或中断服务程序中实现。

使用者可以使用两个明确循环命令：设置优先顺序命令以及明确 EOI 循环命令。设置优先顺序命令使程序设计者能将某一 IRQ 层次设为最低优先，而其它中断层次则根据新设置，符合全嵌套形态。明确 EOI 循环命令则为设置优先顺序命令与明确 EOI 命令的组合。像设置优先顺序命令一样，该命令能将某一 IRQ 层次设为最低优先，而像明确 EOI 命令一样，ISR 中的某一明确中断层次将被清除为 0。因此该命令以一个命令完成两种工作。

3. 中断屏蔽形态

中断屏蔽可通过中断屏蔽寄存器或特殊的屏蔽形态完成。82380PIC 的每一中断层都有一个中断屏蔽寄存器(IMR)。此屏蔽寄存器 IMR 使个别的 IRQ 被屏蔽成为可能。在某一 IRQ 被屏蔽掉后，其中断请求即被禁止。直至屏蔽位清除为止。8 位 IMR 的每位可分别被设置为“1”，使每一中断通道被禁止。其中第 0 位用以屏蔽掉 IRQ0，第 1 位以蔽掉 IRQ1，依此类推。

利用中断屏蔽形态和特殊屏蔽形态合用，在某些情况下，可以通过程序实现优先顺序较低的中断请求能中断优先顺序较高的中断处理程序的执行。

4. 边沿或电平中断触发形态

82380PIC 的每一中断层均可独立设置其中断请求信号采用边沿触发有效还是电平触发有效，可以通过程序定义。如前所述，所有 IRQ 输入均为低电平有效。因此，在定义电平触发时，IRQ 输入信号由高电位换成低电位时产生的信号边沿即为有效沿。定义成电平触发时，中断请求信号只要是低电平就是输入。不过为了避免同一中断请求被处理两次，则中断被处理后，必须在送出 EOI 命令之前，将中断信号清除。或在主 CPU 执行中断服务程序过程中中断执行。同样，为了被主 CPU 响应，不论采用哪一种形式，都要使中断请求输入信号保持至第一 $\overline{\text{INTA}}$周期，否则，最后 PIC 会送出既定的 IRQ7 向量。

5. 中断层串接形态

如前所述，82380PIC 上的任一中断请求线 IRQ 上均可串接一从中断控制器(如 8259A)。在从中断控制外设被服务时，82380PIC 在第一个$\overline{\text{INTA}}$周期时，会将每个与外设有关的向量寄存器的内容(而不是平常的 00H)置于数据总线上。因此，在第一$\overline{\text{INTA}}$周期时，一个外加的锁存器必须借$\overline{\text{INTA}}$状态信号锁住此向量值，并以此选取应服务的外接从中断控制器。此被选取的从中断控制器必须在第二个$\overline{\text{INTA}}$周期时将其中断向量置于数据总线上。

在串接的从中断控制器内的优先顺序要保存时，优先顺序可采用特殊的全嵌套形态，除了下列例外情况之外，特殊的全嵌套形态与正常的全嵌套形态一样：

其一，在从中断控制器外设所提出的中断请求正在被服务时，此从中断控制器外设并未被 82380PIC 的优先顺序电路所隔绝。因此，若紧接从中断控制器又有其他更优先的中断请求发

生，则此一中断请求也会为82380PIC所认可，并进而再中断主CPU80386/80486。在正常的全嵌套形态内，从中断控制器外设中某一外设的中断请求正被服务时，整个从中断控制器即会被锁住。因此，该从中断控制器中其他更优先的中断请求即不能被处理。

其二，在离开中断处理程序时，软件必须检查目前被服务的中断是否是从中断控制器中唯一的中断请求。这可经由送出一不明确EOI命令给从中断控制器，然后再读取其中断服务寄存器(ISR)得知。若已知从中断控制器中已无其他的中断请求，则程序即可同时送出一不明确的EOI命令给对应的82380PIC的中断层。否则EOI即不应该送出。

6. 读取中断状态形态

82380PIC有两种方式可知每一中断层的状态。一种是取样尚未确定的最高优先中断请求，另一种则读取中断状态寄存器的内容。

82380有取样命令，可用以测知尚未被处理完的中断请求中，那一个具有最高优先顺序。使用此命令时，INT输出即不用，或是主CPU的中断响应被禁止。取样命令的另一个作用是可用来扩充优先层次的数目。

取样命令并不支持ICW_2方式，不过，若使用了取样命令，则由于不会有$\overline{INTA}$周期产生，可编程向量寄存器的内容也就无所谓。

除了取样命令外，使用程序也可读取所有的中断寄存器(IRR，ISR及IMR)，以得知82380PIC的现有状态。读取IRR和ISR的内容可以利用读取状态寄存器命令，并通过第三操作命令字完成。而读取IMR的内容值则可简单地由读取寄存器的读操作完成。

四、可编程寄存器

82380PIC每一中断层均含有一组8位的寄存器，以控制其操作。三个中断层的寄存器功能相同，只介绍其中一个。

1. 初始化命令字寄存器(ICW)

在开始工作之前，82380PIC必须设置成一已知状态。因此，每一中断层均有四个初始化命令字寄存器，以设置各种所需的操作形态和状况。这四个寄存器中，除了第二个(ICW_2)是可读的寄存器外其他三个都是只能写入的寄存器。

ICW_1：

该寄存器有三个主要功能：(1)选取IRQ输入的触发形态(边沿或电平)。(2)显示中断层是单独使用或处于串接形态若想要串接形态，则中断层会接受ICW_3寄存器作为串接形态的初始编程用。否则不必接受ICW_3。(3)决定是否ICW_4会被使用。

ICW_2：

该寄存器主要作为与8259A相容用的。其内容并不影响中断层的其它操作。在三个中断层的ICW_2有任一个被写入时，A中断层的IRQ1，5即产生一个中断。而在主CPU80386/80486读取ICW_2寄存器后，该中断请求即自动清除。使用者应设置对应的向量寄存器和以此作为指示器，显示有人想改变内容，需指出，三个ICW_2应使用不同的地址进行读/写操作。

ICW_3：

若用户需连接成外部串接形态，则中断层将接受一个ICW_3用于在串接形态内完成明确的初始编程。其某一位显示那一种中断请求输入有串接一个中断控制器。此一串接会影响中断响应周期时中断向量的产生。

ICW_4：

此寄存器只有在被 ICW_1 中选到时，才会被接受。ICW_4 有两个功能：一是选择自动 EOI 模式或软件 EOI 模式。一是选择特殊嵌套形态是否要与串接形态合用。

2. 操作命令字寄存器(OCW)

一旦由 ICW 启动后，82380PIC 的中断层即操作在既定的全嵌套形态，而且可以随时接受中断请求。不过，每一中断层的操作可以进一步利用 OCW 控制字控制和改变。PIC 共有三个 OCW 寄存器可用。每一个都是 8 位只能写入的寄存器。OCW 所控制的形态及操作包括全嵌套形态，循环优先顺序模式，特殊屏蔽形态，取样形态，EOI 命令及读取状态命令。

OCW_1：

只用于屏蔽操作。与中断屏蔽寄存器有直接关连。主 CPU 可以写入 OCW_1 寄存器控制字，将中断输入允许或禁止。读取事先安排好的屏蔽可由 IMR 寄存器完成。

OCW_2：

用以选取中断结束，自动循环优先顺序和明确循环优先顺序。不同位的组合可以选取与这些操作有关的命令和形态。也即 OCW_2 用于(1)记载要重置某一特别 ISR 位或设置一明确优先顺序的中断层次(0-7)。此一功能可被允许或禁止。(2)选择要执行那一个 EOI 命令(若有的话)，即不明确或明确 EOI。(3)将指定其中一种优先顺序循环操作，即不明确 EOI 循环、自动 EOI 循环和明确 EOI 循环。

OCW_3：

主要控制以下三个操作 ：(1)选择及执行读取状态寄存器命令，即读取中断请求寄存器(IRR)或中断服务寄存器(ISR)。(2)送出取样命令 。若两种功能同时有效，则取样命令会覆盖读取寄存器命令。(3)设置或清除特殊屏蔽形态。

取样/中断请求/处理状态寄存器

可读取的 8 位寄存器内具有多种功能。由 OCW_3 所送出的命令不同而定，该寄存器的内容反映该命令执行的结果，即在取样命令时，该寄存器含有要求服务的最高优先层次的二进制码；在读出 IRR 命令时，此寄存器的内容即显示目前尚在等待的中断请求；在读取 ISR 命令时，该寄存器即指明正被服务的中断层次。

4. 中断屏蔽寄存器(IMR)

该 8 位寄存器是一个只能读取的寄存器，它的内容指明该中断层被屏蔽掉的所有中断请求。

5. 中断向量寄存器(VR)

每一中断请求输入对应有一个 8 位可读写编程的中断向量寄存器。在 $\overline{INTA}$ 周期时，中断向量寄存器的内容被置于数据总线上。

五、82380 PIC 初始化编程

82380PIC 的初始化编程可由 ICW 初始化命令字加以编程设计。ICW 控制字由主 CPU80386/80486 以循环方式逐一送出，用以将 PIC 的中断层设置成某一已知的起始状态。OCW 命令则用以改变及控制 82380PIC 的操作。

A、B 和 C 三个中断层的编程如图8-24所示即为每一个中断层的初始化编程流程图。

需要指出的是，一旦初值设置程序执行后，有 ICW 的内容更改，则整个 ICW 的系列即必须重新设置一次。而不能只改变个别的 ICW。如前所述，只要三个 ICW_2 中有一个被写入，则 A 中层次的 IRQ1，5＃中断请求即有效。由于三个 ICW_2 共享同一个 A 中断层，故程序必须将三

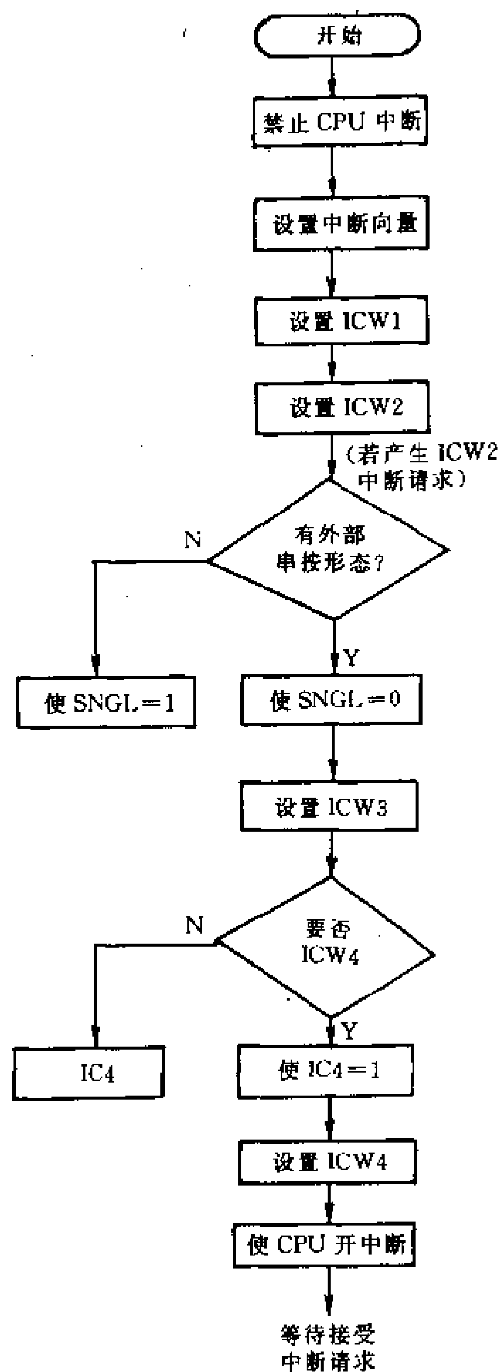


图 8-24　82380PIC 初始化编程流程图

个 ICW_2 都进行读出。

另外，当 ICW 控制字设置好后，每一个中断控制操作即可经由写入 ICW 命令字加以改变。

OCW 命令字的写入，无一定的写入顺序，任何 OCW 命令字均可在任意时刻写入。

由于 IRR 与 ISR 状态的读取以及取样指令的结果都呈现在同一只能读取的状态寄存器，因此，在实际读取取样/中断请求 /服务状态寄存器之前，程序必须先送出一特殊的读取状态/取样命令。此命令可借着将所需要的控制字写入 OCW_3 指明。如前所述，在取样命令与状态读取命令同时作用时，状态寄存器最后含有取样命令的结果。

除了上述的读取命令外，中断屏蔽寄存器(IMR)也可以读取。在读取时，该寄存器会反映事先写入的 OCW_1 的内容，而 OCW_1 则含有那些中断请求目前被禁止的信息。

为了实际进行初始化编程的需要，现将初始设置控制字组和操作控制字组的格式介绍如下：即图 8-25(1～11)

第一初始值设置控制命令字(ICW_1)格式：

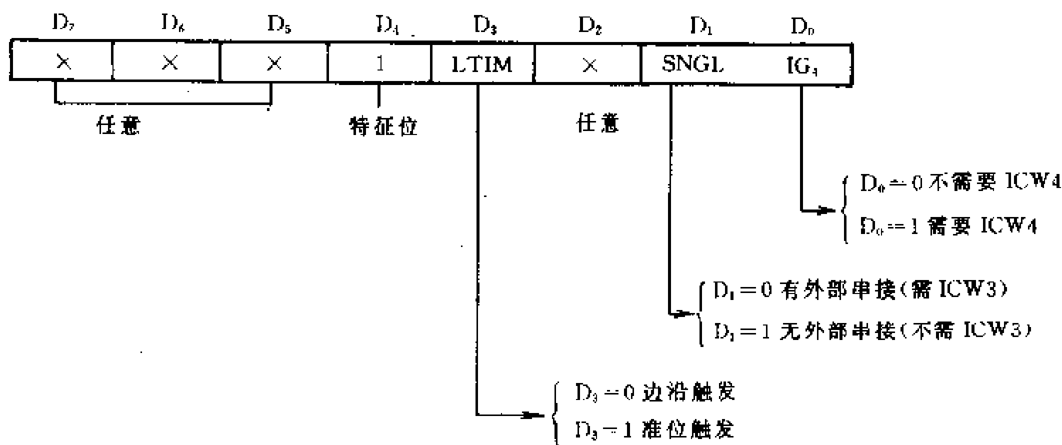


图 8-25(1)

第二初始值设置控制命令字(ICW_2)格式：

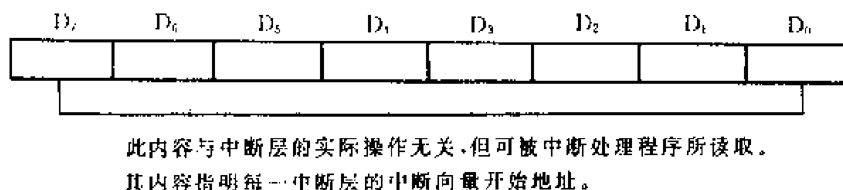
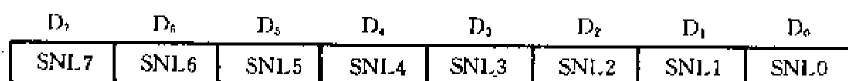


图 8-25(2)

第三初始值设置控制命令字(ICW_3)格式：



SNL＝1　对应中断请求位有串接从中断控制器

SNL＝0 对应位中断请求位无串接从中断控制器

图 8-25(3)

第四初始值设置控制命令字(ICW_4)格式：

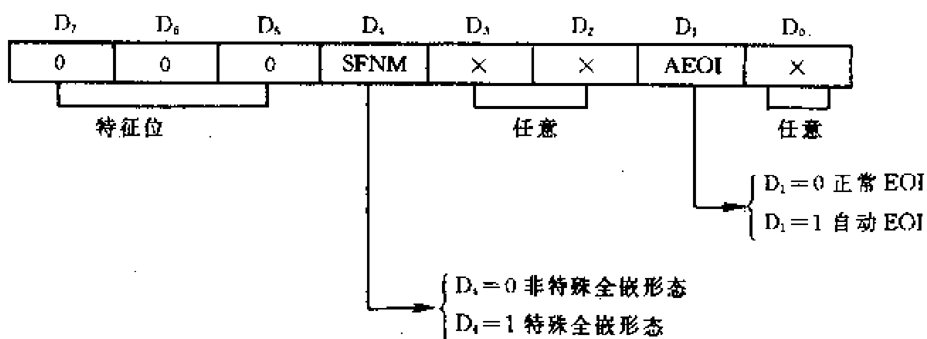
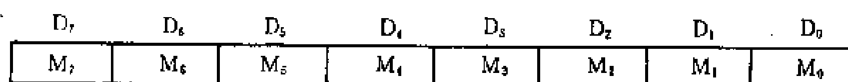


图 8-25(4)

第一工作控制命令字(OCW_1)格式：



$M_i=1$ 屏蔽设置为1，禁止中断

$M_i=0$ 屏蔽设置为0，允许中断

图 8-25(5)

第二工作控制命令字(OCW_2)格式：

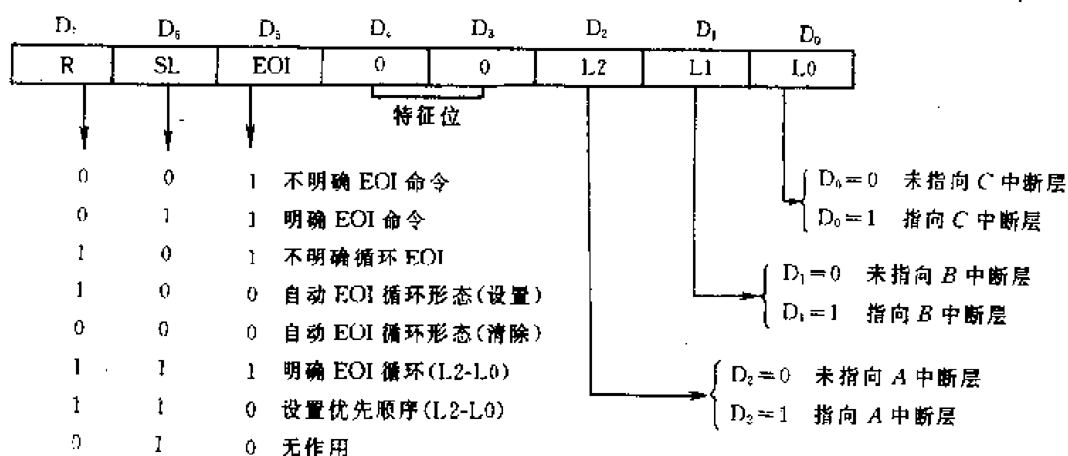


图 8-25(6)

第三工作控制命令字(OCW_3)格式：

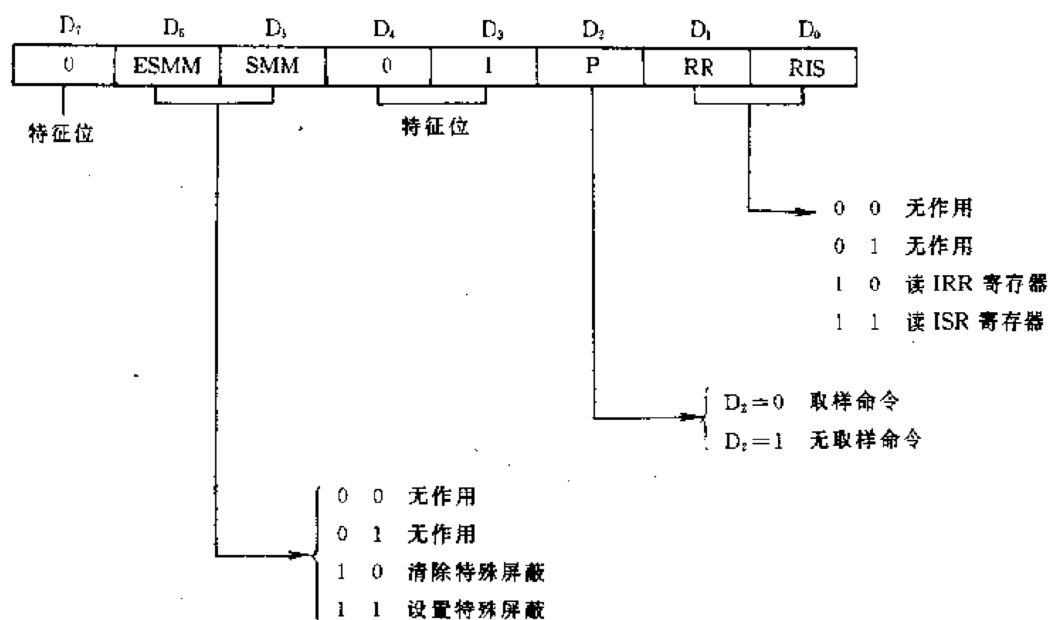


图 8-25(7)

取样/中断请求/处理中断状态寄存器中的命令字格式：

取样命令字格式：

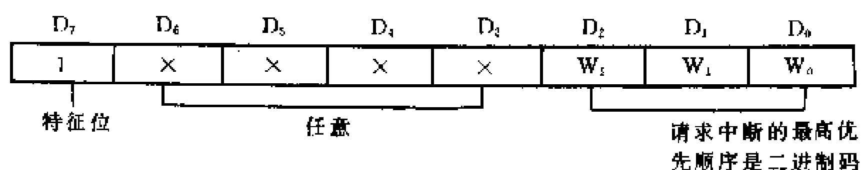
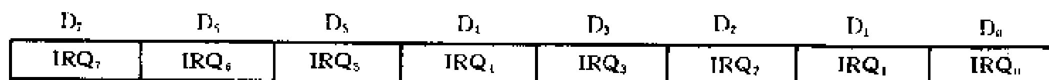


图 8-25(8)

中断请求状态：

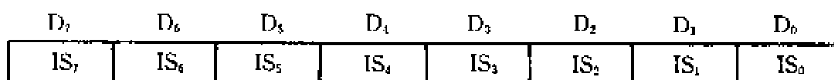


$IRQ_i=0$ 无中断请求

$IRQ_i=1$ 中断请求在等着

图 8-25(9)

处理中断状态：



$IS_i=0$ 表示中断请求 IRQ 已处理

$IS_i=1$ 表示中断请求 IRQ 正在被处理

图 8-25(10)

中断向量寄存器(VR)：

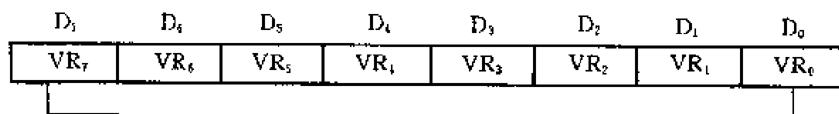


图 8-25(11)

思考题与习题

1. 什么叫中断？
2. 简述中断的处理过程。
3. 中断系统通常应具备哪些功能？
4. 8086/8088 中各类中断的优先级别是如何排列的？
5. 8086/8088 的中断共分为哪几种？
6. 何为中断向量和中断向量表？
7. 8086/8088 CPU 响应可屏蔽中断 INTR 的条件是什么？

8. CPU 响应外部中断的周期里为何要连续产生两个$\overline{INTA}$信号？

9. 应用程序在修改中断向量时为何要关中断？

10. 选择题

(1)CPU 响应两个硬件中断 INTR 和 NMI 时相同的必要条件是(　)。

A. 允许中断　　B. 当前指令执行结束

C. 总线空闲　　D. 当前访存操作结束

(2)当单步跟踪标志 TF=1 引发的单步中断程序却能连续执行，其原因是(　)。

A. TF=0　　B. 中断允许

C. 现场已进栈保存　　D. 与 TF 无关

(3)8259A 在上电初始化后，为防止与操作命令字 OCW 冲突，ICW 命令流程的原则是(　)。

(4)设 8259A 当前最高优先级为 IR_5，若想使该请求变为下一循环的最低优先级，输出 OCW_2 的数据格式是(　)。

A. 10100101　　B. 11100000

C. 01100101　　D. 10100000

(5)设 8259A 当前最高优先级为 IR_5，若想使下一循环请求中最低优先级为 IR_2，则输出 OCW_2 的数据格式是(　)。

A. 10100010　　B. 01100010

C. 11100010　　D. 11000010

(6)在两片 8259A 级联的中断系统，主片的第五级 IR_5 作为从片的中断请求则初始化主·从片时，ICW_3 的数据格式分别是(　)。

A. 05H 和 20H　　B. 50H 和 02H

C. 02H 和 50H　　D. 20H 和 05H

(7)PC/XT 机采用向量中断方式处理 8 级外中断，中断号依次为 08H～0FH。在 RAM 0：2CH 单元开始依次存放 23H，FFH，00H 和 F0H 四个字节，问该向量对应的中断号和中断程序入口是(　)。

A. 0CH 和 23FF:00F0H　　B. 0BH 和 F000:FF23H

C. 0BH 和 00F0:23FFH　　D. 0CH 和 F000:FF23H

E. 0CH 和 00F0:23FFH　　F. 0BH 和 F000:23FFH

(8)在不改变任何硬件的条件下，欲使 PC 系统上电后 8259A 进入查询方式，应用程序入口的充分必要条件是(　)。

A. 关中断　　B. 重新执行初始化

C. 输出 OCW_3 查询位　　D. A，B，C 同时成立

(9)8259A 中断请求选择边沿触发，通常也要求有足够的脉冲宽度，其原因是(　)。

A. 能可靠锁存请求　　B. 防止噪声尖峰产生中断

C. 芯片电气性能要求　　D. A，B 同时成立

第九章　输入/输出接口

随着微型计算机应用越来越广泛，外围设备的种类越来越多，各类设备的组成和工作原理差异很大，与微机之间的连接和传输数据的方式也很不相同。特别是，当微机应用在数据采集和处理系统及过程控制系统时，与微机连接的常常是一些专用设备，对数据传输和处理的实时性要求很高。所以，微型计算机的输入输出是最具有多样性和复杂性的问题。

本章将讨论并行接口、串行接口、定时/计数控制、数/模与模/数接口的接口技术及其应用，通过对典型输入输出接口芯片使用的讨论，力图使读者对输入输出接口技术有一个清楚的了解，并能运用于具体的工程实际之中。

第一节　并行数据通信接口技术

一、并行通信与并行接口

（一）并行通信

并行通信是把一个字的各位各用一条线同时进行传输，传输速度快，信息率高，但它比串行通信所用的电缆多，因此，并行通信常用在传输距离短（几米至几十米），数据传输率要求较高的场合。

（二）并行接口

实现与外设并行通信的接口就是并行接口。一个并行接口可设计为只作为输出接口，如一个并行接口连一台打印机；还可设计为只作为输入接口，如一个并行接口连接卡片读入机。另外，还可以设计成既作为输入又作为输出的接口，它可以用两种方法实现，一种是利用同一个接口中的两个通路，一个作输入通路，一个作输出通路；另一种是用一个双向通路，既作为输入又作为输出。前一种是用在主机需要同时输入和输出的情况，如此接口既接纸带读入机，又接纸带穿孔机。后一种方法是用在输入输出操作并不同时进行的主机与外设之间，如连接两台磁盘驱动器。

典型的并行接口电路如图 9-1 所示，其具体过程可分为：

1. 并行接口的输入过程

外设首先将数据送给接口，并使状态线“数据输入准备好”成为高电平。接口把数据接收到数据输入缓冲寄存器的同时，使“数据输入回答”线变为高电平，作为对外设的响应。外设接到此信号，便撤除数据和“数据输入准备好”信号。数据到达接口中后，接口会在状态寄存器中设置“输入准备好”状态位，以便供 CPU 对其进行查询，接口也可以在此时向 CPU 发一个中断请求。所以，CPU 既可以用软件查询方式，也可以用中断方式来设法读取接口中的数据。CPU 从并行接口中读取数据后，接口会自动清除状态寄存器中的“输入准备好”状态位，并且使数据总线处于高阻状态。此后，又可以开始下一个输入过程。

2. 并行接口的输出过程

每当外设从接口取走一个数据之后，接口就会将状态寄存器中的“输出准备好”状态位置

"1"。以表示CPU当前可以往接口中输出数据，这个状态位可供CPU进行查询。此时，接口也可以向CPU发一个中断请求。所以CPU既可以用软件查询方式，也可以用中断方式设法往接口中输出一个数据。当CPU输出的数据到达接口的输出缓冲寄存器中后，接口会自动清除"输出准备好"状态位，并且将数据送往外设，同时，接口往外设发送一个"驱动信号"来启动外设接收数据。外设被启动后，开始接收数据，并往接口发一个"数据输出回答"信号。接口收到此信号，便将状态寄存器中的"输出准备好"状态位重新置"1"，以便CPU输出下一个数据。

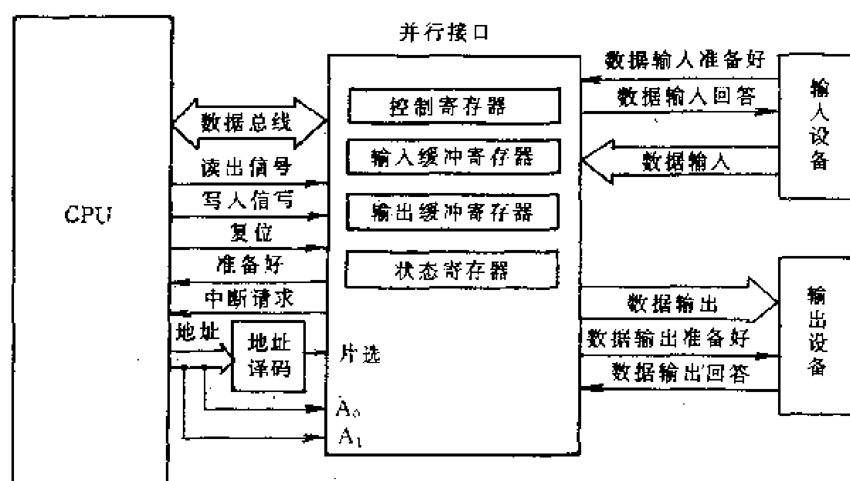


图 9-1 并行接口电路示意图

二、可编程并行通信接口芯片 8255A

(一) 8255A 的内部结构

8255A 组成方框图见图 9-2，它由三部分组成：外设接口部分(通道 A、B、C)，内部逻辑部分(A 组和 B 组控制电路)和 CPU 接口部分(数据总线缓冲器，读/写控制逻辑)。

(1) 外设接口部分

8255A 有 3 个 8 位数据通道 A、B、C，这 3 个通道在功能上各有特点，都可编程设置为输入或输出。

通道 A 具有一个 8 位数据输入锁存器和一个 8 位数据输出锁存器/缓冲器用来传送数据。作输入端口或输出端口时数据均受到锁存。

通道 B 具有一个 8 位数据输入缓冲器和一个 8 位数据输出锁存器/缓冲器，也用来传送数据作输入端口时不会对数据进行锁存，而用作输出端口时数据受到锁存。

通道 C 具有一个 8 位数据输入缓冲器和一个 8 位数据输出锁存器/缓冲器，一般作为控制或状态信息端口，可分成两个 4 位端口(高位口和低位口)，分别和 A 通道、B 通道配合使用。当 CPU 与外设连接不需要联络控制线时，C 通道可以和 A、B 通道一样作为输入或输出的数据通道。

(2) 内部逻辑

有 A、B 两组控制电路。每组控制电路从读/写控制逻辑接收各种命令，从内部数据总线接收控制字并发出相应命令到各自的通道，控制各个通道的工作方式。还可以根据 CPU 的命令字对通道 C 的每一位按位置位或复位。

A 组控制电路控制通道 A 和通道 C 的高 4 位(PC_7～PC_4)。

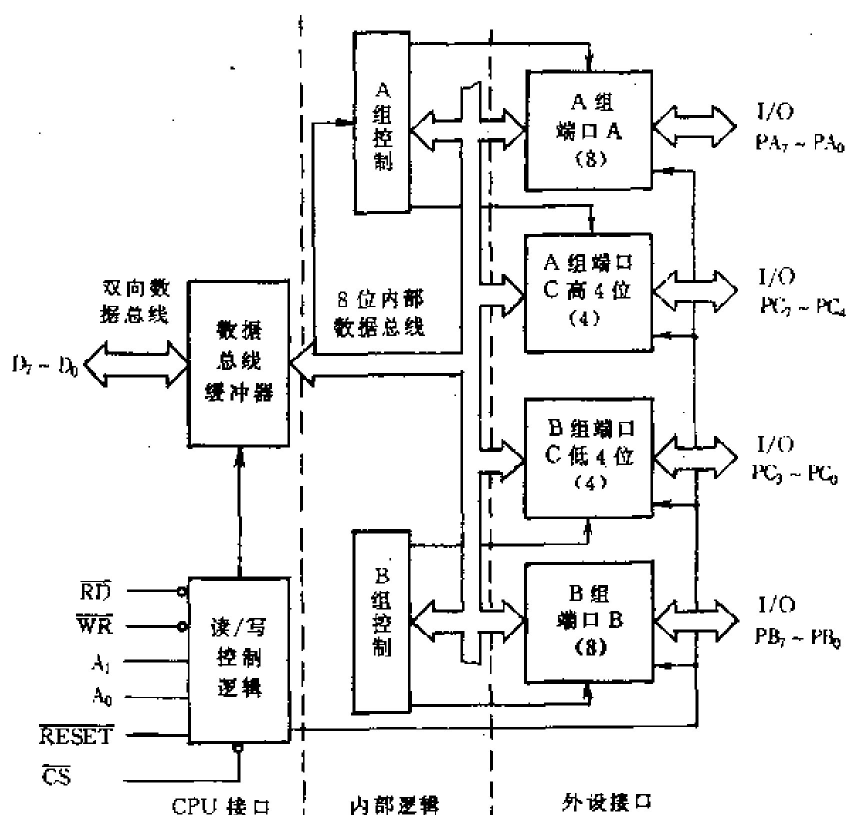


图 9-2 8255A 内部结构图

B 组控制电路控制通道 B 和通道 C 的低 4 位(PC_3～PC_0)。

(3) CPU 接口部分

这部分包括数据总线缓冲器和读/写控制逻辑。

数据总线缓冲器是 8255A 与 CPU 数据总线的接口，是 8 位双向三态缓冲器，由读/写控制逻辑实施三态控制。所有数据的输入和输出，以及 CPU 写入 8255A 的控制字，从 8255A 读出的外设状态信息，都是通过这个缓冲器传送的。读/写控制逻辑接收有关控制信号$\overline{RESET}$、$\overline{WR}$、$\overline{RD}$和地址总线的 A_1、A_0(8086 系统中可为 A_2、A_1)、地址译码信号$\overline{CS}$，对 A 组和 B 组控制电路实施控制，管理内部和外部数据、状态或控制字的传送。

(二)8255A 芯片的引脚及其功能

图 9-3 是 8255A 的芯片引脚信号，其引脚信号分为：

(1) 和外设相连的

PA_7～PA_0：A 口数据信号线

PB_7～PB_0：B 口数据信号线

PC_7～PC_0：C 口数据信号线

(2)和 CPU 相连的

$\overline{RESET}$：复位信号，低电平有效。当$\overline{RESET}$信号来到时，所有内部寄存器都被清除，同时 3 个数据端口被自动置为输入端口。

D_7～D_0：它们是 8255A 的数据线，和系统数据总线相连。

$\overline{CS}$：片选信号，低电平有效。在一个系统中，一般根据全部接口芯片来分配若干低位地址

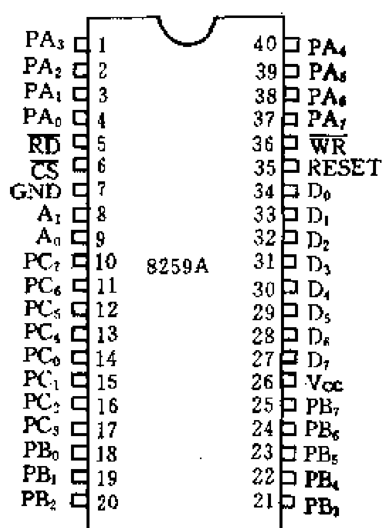


图 9-3　8255A 芯片引脚图

(比如 A_5、A_4、A_3)组成各种芯片选择码，当这几位地址组成某一个低电平，于是，8255A 被选中。只有当$\overline{CS}$有效时，读信号$\overline{RD}$和写信号$\overline{WR}$才对 8255A 进行读写。

$\overline{RD}$：读信号，低电有效。当$\overline{RD}$有效时，CPU 可以从 8255A 中读取数据。

$\overline{WR}$：写信号，低电平有效。当$\overline{WR}$有效时，CPU 可以往 8255A 中写入控制字或数据。

A1、A0：端口选择信号。8255A 内部有 3 个数据端口和 1 个控制端口，共 4 个端口。规定当 A1、A0：为 00 时，选中 A 端口；为 01 时，选中 B 端口；为 10 时，选中 C 端口；为 11 时，选中控制口。

(3)电源和地

V_{CC}：+5V 电源

GND：地线

(三) 8255A 的编程

1. 8255A 的控制字

(1) 8255A 的控制字

图 9-4(a)列出了各位的作用。最高位 D_7 必须为"1"，是方式控制字的特征位。当用输出指令将方式控制字写入 8255A 后，它被分存于 A、B 两组控制寄存器中。D_6～D_3 控制 A 口及 C 口高 4 位(合称 A 组)的工作方式以及输入还是输出。D_2～D_0 控制 B 口及 C 口低 4 位(合称 B 组)的工作方式及输入或输出。

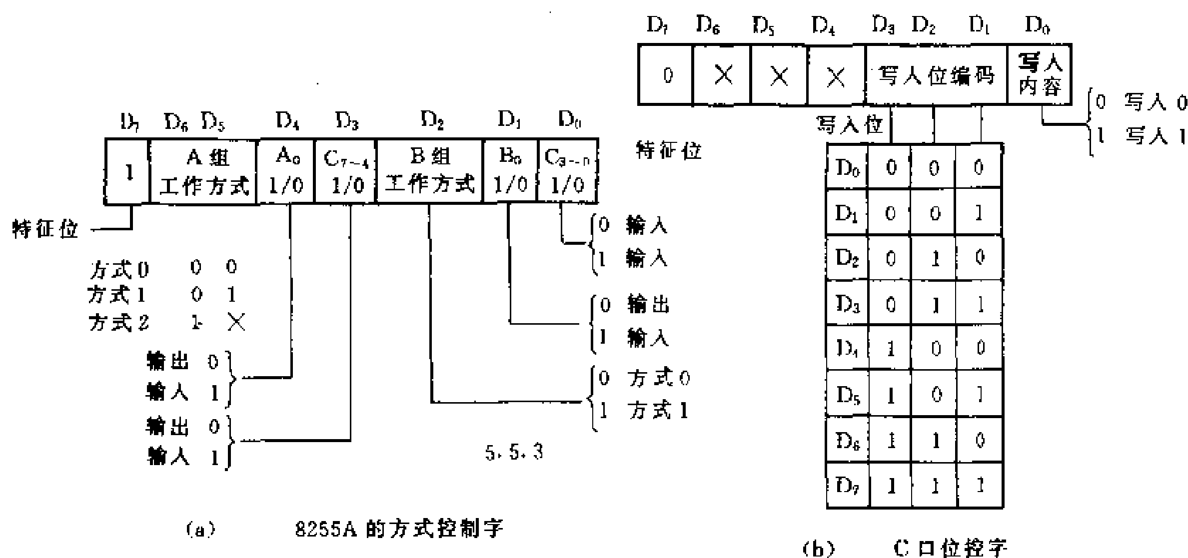


图 9-4　8255A 的方式控制字和 C 口的位控字

(2) C 口的位控字

8255A 的 C 口具有位控功能，即允许 CPU 用输出指令单独对 C 口的某一位写"1"或"0"，格式如图 9-4(b)。这是通过向 8255A 的控制寄存器写入(注意不是直接对 C 口写入)一个位控字来实现的。后面将讲到 A 口及 B 口的中断允许就是通过位控字写入 C 口的。

2. 8255A 的三种工作方式

(1) 方式 0

方式 0,即基本输入或输出方式,这种方式下,无固定的 I/O 联络信号。此方式下 A 口、B 口、C 口的高 4 位和低 4 位可以分别设置成输入或输出。

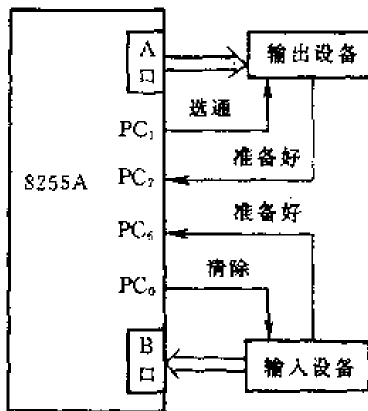


图 9-5　方式 0 下 I/O 的联络信号

图 9-5 是一个方式 0 下利用 C 口作为联络信号的 I/O 电路。此例中将 8255A 编程为如下状态:A 口输出,B 口输入,C 口高 4 位为输入,现仅用 PC_7、PC_6 两位输入外设的状态;C 口低 4 位为输出,用 PC_1、PC_0 输出选通及清除信号。对 8255A 写入的控制字为:8AH=10001010B。

工作中,在给输出设备送数前,先通过 PC_7 查询设备状态,若准备好再从 A 口送出数据,然后用 PC_1 发选通信号使使输出设备接收数据。从输入设备取数前,也先通过 PC_6 查询设备状态,准备好后,再从 B 口读入数据 ,然后从 PC_0 发清除信号,以便输入后续字节。

和后面讲到的方式 1 的选通 I/O 相比,方式 0 的联络信号线可由用户自行安排,只能用于无条件传送和查询传送,不能实现中断传送。

(2) 方式 1

方式 1 即字节选通 I/O 方式,此时,接口和外设之间需要联络信号进行协调,只有 A 口和 B 口可工作于方式 1。此时 C 口的某些线被固定作为 A 口或 B 口与外设之间的联络信号线,其余的线只能定义为基本 I/O,即只能工作于方式 0。

a. 方式 1 输入

方式 1 输入时,各端口线的功能见图 9-6。

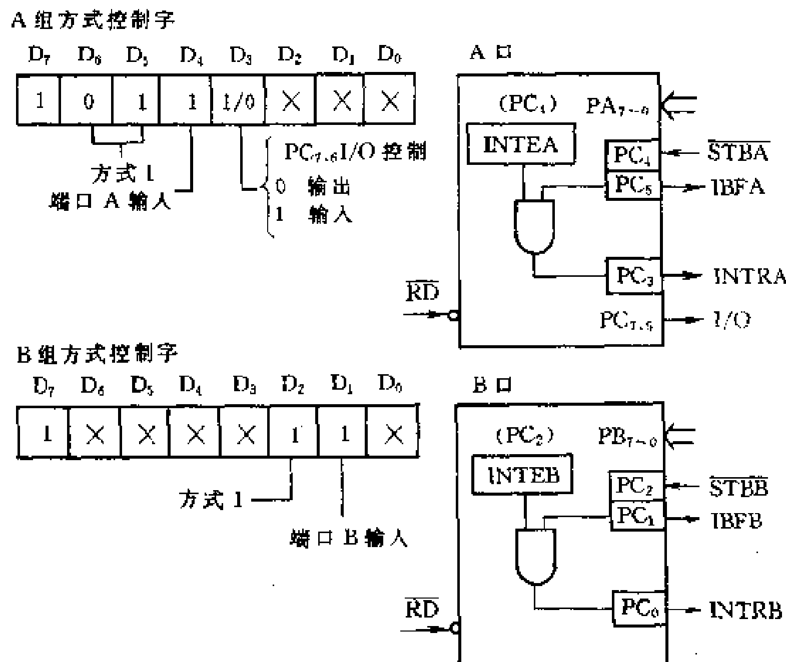


图 9-6　8255A 方式 1 输入时的控制字及信号

A 口工作于方式 1 输入，固定用 $PC_5 \sim PC_3$ 作联络信号线；B 口工作于方式 1 输入，固定用 $PC_2 \sim PC_0$ 作联络信号线。各信号的作用说明如下：

$\overline{STB}$(STROBE)选通信号，输入，低电平有效。它将外设的信号输入 8255A 的锁存器。接 PC_4、$\overline{STB_A}$接 PC_2。

IBF(INPUT BUFFER FULL)输入缓冲器满信号，输出，高电平有效，通知外设送来的数据已被接收，由$\overline{STB}$信号的前沿产生。当 CPU 用输入指令读走数据后，此信号被清除。A，B 两口的 IBF 信号分别由 PC_5 及 PC_1 提供。

INTR(INTERRUPT REQUEST)中断请求信号，输出，高电平有效，在中断允许 INTE=1，且 IBF=1 的条件下，由$\overline{STB}$信号的后沿产生，可接至中断管理电路 8259A 作中断请求 CPU 响应中断后在服务程序中读走数据时，由$\overline{RD}$信号将其清除。INTRA 由 PC_3 提供。INTRB 由 PC_0 提供。

INTE(INTERRUPT ENABLE)中断允许位，INTE=1 允许中断，INTE=0 禁止中断，可事先用位控方式写入。$INTE_A$ 写入 PC_4，$INTE_B$ 写入 PC_2。

在这种情况下，若读入 C 口状态，其各位所表示的意义如图 9-7(a)。

其中最高二位是当前 I/O 的数据，两个中断允许位 D_4、D_2 是事先用位控字写入的内容，其余各位都反映实时状态。

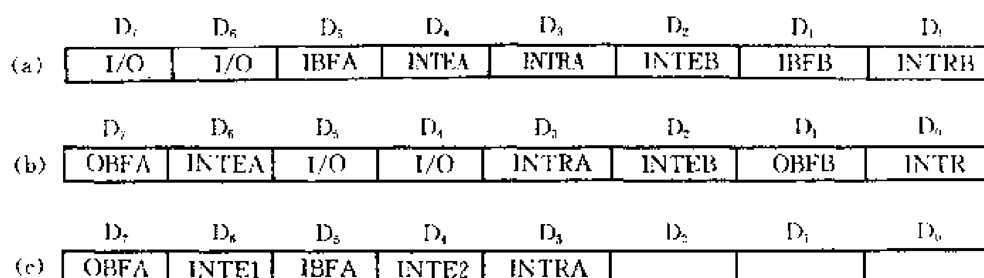


图 9-7 不同方式下 C 口状态各位含义

b. 方式 1 输出

方式 1 输出时，各端口线的功能见图 9-8。

A 口工作于方式 1 输出，所用的联络信号线为 PC_7、PC_6 和 PC_3，而 B 口工作于方式 1 输出时，使用 $PC_2 \sim PC_0$ 作其联络信号线。各联络信号的作用如下所述：

$\overline{OBF}$(OUTPUT BUFFER FULL) 输出缓冲器满，低电平有效。当 CPU 给端口写入一个字节数据时，由$\overline{WR}$信号上升沿使$\overline{OBF}$有效，通知外设可以取数据。A、B 两口的$\overline{OBF}$信号分别由 PC_7 及 PC_1 提供。

$\overline{ACK}$(ACKNOWLEDGE) 应答信号，低电平有效。当外设得知$\overline{OBF}$信号，取数据时，要发出 ACK 信号选通，取走数据并清除$\overline{OBF}$。A、B 两口的$\overline{ACK}$信号分别由 PC_6 及 PC_2 提供。

INTR 中断请求信号，其作用及引出端都和方式 1 输入时相同，但由$\overline{ACK}$信号的后沿在 INTE=1 且$\overline{OBF}$=1 的条件下产生。若 CPU 响应中断，往该端口写入一字节数据，其$\overline{WR}$信号将清除 INTR。

INTE 中断允许位，方式 1 输出组态下，A 口的中断允许位写入 PC_6。而 B 口的仍写入 PC_2。此种组态下，读入 C 口状态各位含义如图 9-7(b)。

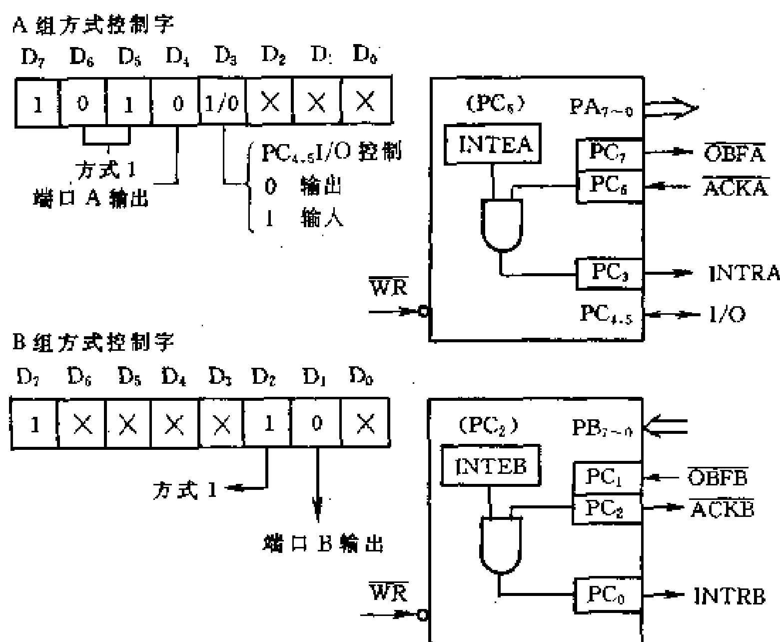


图 9-8 8255A 方式 1 输出时的控制字及信号

(3) 方式 2

方式 2 为双向 I/O 方式，即同一端口的 I/O 线既可以作为输入也可以作为输出。只有 A 口可以工作于方式 2，此时 C 口有 5 条线被固定为 A 口和外设之间的联络信号线。C 口余下的 3 条线可以作为 B 口方式 1 下的联络线，也可以和 B 口一起成为方式 0 的 I/O 线。如图 9-9 是 8255A 工作方式 2 下的控制字及信号组合，说明如下：

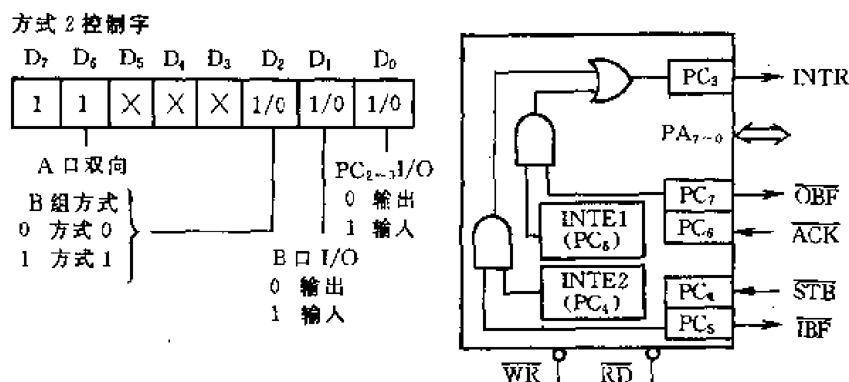


图 9-9 8255A 方式 2 时的控制字及信号

1. A 口以方式 2 输出的数据，仅在 $\overline{ACK}$ 信号有效时才出现在 A 口的数据线 $PA_7 \sim PA_0$ 上，否则这些线都呈高阻态。因此外设产生的应答信号 $\overline{ACK}$ 应是个负方波，其前沿让 8255A 输出数据，其后沿使数据锁存于外设中。

2. A 口方式 2 的 I/O 共用一个中断请求 INTR 信号（由 PC_3 提供），但中断允许位仍是各自的，输出的中断允许位写入 PC_6（图中称 $INTE_1$），输入的中断允许位写入 PC_4（图中称 $INTE_2$）。

3. A 口在方式 2 时，B 组（$PB_7 \sim PB_0$ 及 $PC_2 \sim PC_0$）仍可有两种工作方式选择：若选方式 0，则 $PC_2 \sim PC_0$ 可独立选择为 I/O（由控制字的 D_0 决定）；若选方式 1，则 $PC_2 \sim PC_0$ 为 B 口联

络线(参看方式 1 中 B 组的组态),这时控制字的 D_0 失去意义。

4. 在方式 2 下读端口 C 所得各位状态的意义如图 9-7(c)。

其中 D_7~D_3 属于 A 组,除二位中断允许信号(D_6、D_4)是用对 C 口的位控字写入的，其他均实时反映 A 口的工作状态。D_2~D_0 位的含义和 B 口工作方式有关，若 B 口工作于方式 0，则其为 C 口 PC_2~PC_0 的 I/O 线上的数据;若 B 口工作于方式 1 输入,则这三位含义与前面在方式 1 输入组态中介绍的状态一样;若 B 口工作于方式 1 输出,则该三位含义同方式 1 输出组态相同。

(四) 8255A 的应用举例

在这个例子里,8255A 的通道 A 与一个全译码键盘连接,而通道 B 和一个视频显示器连接。如图 9-10(a)所示。所以通道 A 为方式 1 输入而通道 B 为方式 1 输出,方式控制字 MODE＝0B4H。

在这个例里采用了中断查询,中断服务程序的框图如图 9-10(b)所示。查询的目的，是要在所有可能产生中断的中断源中查看是哪一个外部设备产生的中断。这就有一个查询的次序问题。这实际上就是由软件确定中断级别,即先查询的设备被认为是中断级别最高的设备,如此例中的通道 B,这就需要根据外部设备要求服务的轻重缓急来确定查询的先后次序。

8255A 方式 1 中断的查询是在中断后通过读通道 C 状态字的 INTRA(D_3)和 INTRB(D_0)位来确定的。这就要用到屏蔽字,即把要查询的位以外的所有其它状态位都屏蔽掉。INTRA 的屏蔽字 $MASK_A$＝10H,$INTR_B$ 的屏蔽字 $MASK_B$＝01H。在对 8255 初始化时,还需要分别使通道 A 和 B 的中断开放,否则 8255A 将不能产生中断。使通道 A 中断开放用 PC_4 置位控制字 INTEAS＝09H,通道 B 开放用 PC_2 置位控制字 INTEBS ＝05H。程序如下:

```
                                        ;方式 1 应用举例
MODE:       EQU     0B4H                ; 8255A 方式控制字
INTEAS:     EQU     09H                 ; 通道 A 中断允许控制字 PC4=1
INTEBS:     EQU     05H                 ; 通道 B 中断允许控制字 PC2=1
MASKA:      EQU     10H                 ; 保留 INTRA 状态位的屏蔽字
MASKB:      EQU     01H                 ; 保留 INTRB 状态位的屏蔽字
START:      MOV     AL, MODE
            OUT     CNTRL, AL           ; 设置 8255A 工作方式
            MOV     AL, INTEAS
            OUT     CNTRL, AL           ; 通道 A 中断开放
            MOV     AL, INTEBS
            OUT     CNTRL, AL           ; 通道 B 中断开放
            STI                         ; 系统中断开放
                    ⋮
```

8255A 中断服务程序如下:

```
INTSVC: PUSHF                   ;保留 AF 状态
        IN      AL, PORTC       ;从通道 C 读状态字
        MOV     BL, AL          ;状态字保留在 BL 中
```

```
AND     AL, MASKB ;屏蔽掉除 INTRB 以外的其它位
JNZ     PBSVC       ;是通道 B 中断,转显示器服务程序 PBSVC
MOV     AL, BL      ;不是道 B 中断,恢复状态字
AND     AL, MASKA ;保留 INTRA 位
JNZ     PASVC       ;是通道 A 中断吗? 是,转键盘服务程序 PASVC
        ⋮
POPF
IRET
```


(a) 8255A 与键盘和显示器接口　　　　(b) 中断服务程序流程图

图 9-10　8255A 与键盘和显示器接口

第二节　串行数据通信接口技术

一、串行通信与串行接口

（一）串行通信

CPU 与外部的信息交换称为通信(COMMUNICATION),按与外部数据传输的形式,通信方式有两种:

并行通信——数据的所有位同时传送

串行通信——数据逐位顺序传送

上一节介绍了并行通信及其接口电路,这一节讨论串行通信及其接口电路。串行通信指

与外设之间的数据传送是逐位依次传输的，每一位数据占据一个固定的时间长度。这种情况只要少数几条线就可以在系统间交换信息。特别适合于计算机与计算机，计算机与外设之间的远距离通信，但串行通信的速度比较慢。

（二）串行接口

串行接口有许多种类，典型的串行接口如图 9-11 所示，它包括四个主要寄存器，即控制寄存器、状态寄存器、数据输入寄存器及数据输出寄存器。

控制寄存器用来接收 CPU 送给此接口的各种控制信息，而控制信息决定接口的工作方式。状态寄存器的各位叫状态位，每一个状态位都可以用来指示传输过程中的某一种错误或者当前传输状态。数据输入寄存器总是和串行输入/并行输出移位寄存器配对使用的。在输入过程中，数据一位一位从外部设备进入接口的移位寄存器，当接收完一个字符以后，数据就从移位寄存器送到数据输入寄存器，再等待 CPU 来取走。输出的情况和输入过程类似，在输出过程中，数据输出寄存器和并行输入/串行输出移位寄存器配对使用。当 CPU 往数据输出寄存器中输出一个数据后，数据便传输到移位寄存器，然后一位一位地通过输出线送到外设。

CPU 可以访问串行接口中的四个主要寄存器。从原则上说，对这四个寄存器可以通过不同的地址访问，不过，因为控制寄存器和数据输出寄存器是只写的，状态寄存器和数据输入寄存器是只读的，所以，可以用读信号和写信号来区分这两组寄存器，再用一位地址来区分两个只读寄存器或两个只写寄存器。

由于这种串行接口控制寄存器的参数是可以用程序来修改的，所以称作可编程串行接口。

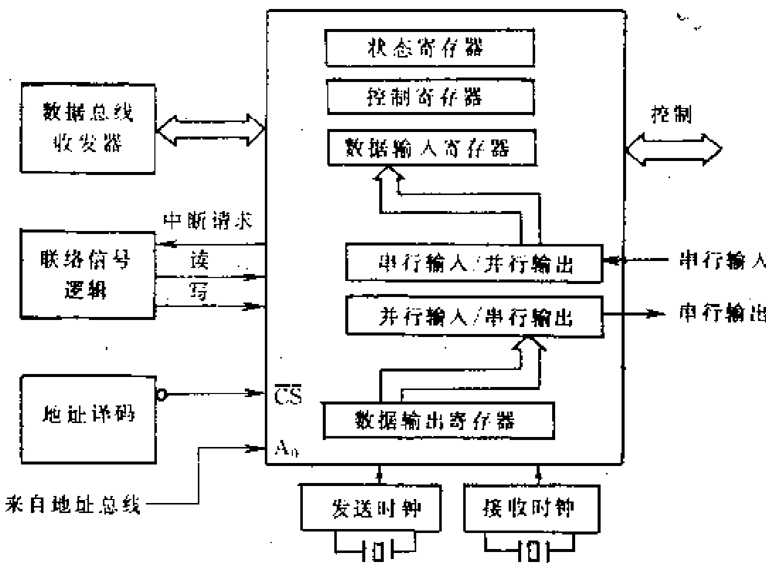


图 9-11　串行接口典型结构

（三）有关串行通信的几个问题。

1. 串行通信的连接方式

串行通信有三种连接方式：单工（Simplex）方式，半双工（Half-Duplex）和全双工（Full-Duplex）方式。

（1）单工方式

这种方式只允许数据按照一个固定的方向传送。采用该方式时，已经确定了通信两点中的一点为接收端，另一点为发送端。这种确定是不可更改的，如图 9-12(a)所示。在参加通信的

a、b 两端中，a 只能为接收器，b 只能为发送器。反之，不行。

(2) 半双工方式

参加通信的 a、b 两端均具备接收或发送数据能力。由于 a、b 是由一条信道相连，故在某一特定的时刻，a、b 的传输方式是明确的，b 发 a 收或 a 发 b 收。决不允许 a 或 b 在同一时刻既发又收，如图 9-12(b)所示。

(3) 全双工方式

由图 9-12(c)可见，全双工方式中是由两条信道将 a、b 两端连接的。从而克服了单工或半双工方式带来的 a、b 两端不能同时既发又收的缺点。为了实现全双工传输的功能，a 端和 b 端必须分别具备一套完全独立的接收器和发送器。

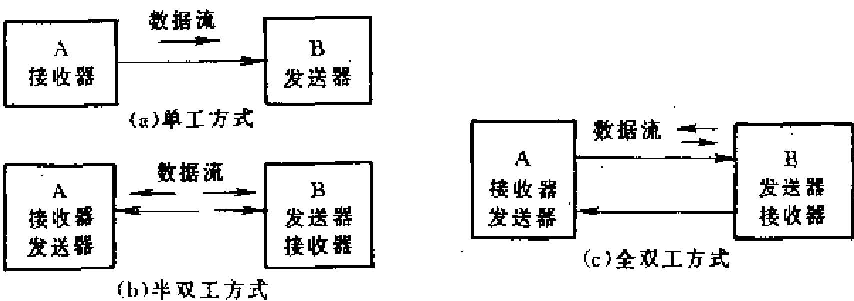


图 9-12 串行通信三种连接示意图

2. 串行通信协议。

为使通信能顺利进行，发送方和接收方要共同遵守一些基本通信规程(protocol)，这些规程在计算机网络中又称为协议，它包括：收发双方的同步方式、传输控制步骤、差错检验方式、数据编码、数据传输速率、通信报文的格式及控制字符的定义等等。目前有两类通信协议：异步通信协议和同步通信协议。

(1) 异步通信协议。

异步通信所采用的数据格式是以一组可变“位数”的数组组成。第一位称起始位，它的宽度为 1bit，低电平；接着传送一个数据 5～8bit，以高电平为“1”，低电平为“0”；也可有一位奇偶校验位；后是停止位，宽度可以是 1bit、1.5bit 或 2bit，在两个数据位之间可有空闲位。异步通信的数据格式见图 9-13。

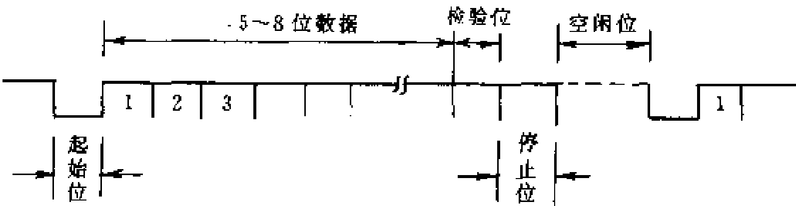


图 9-13 异步通信的数据格式

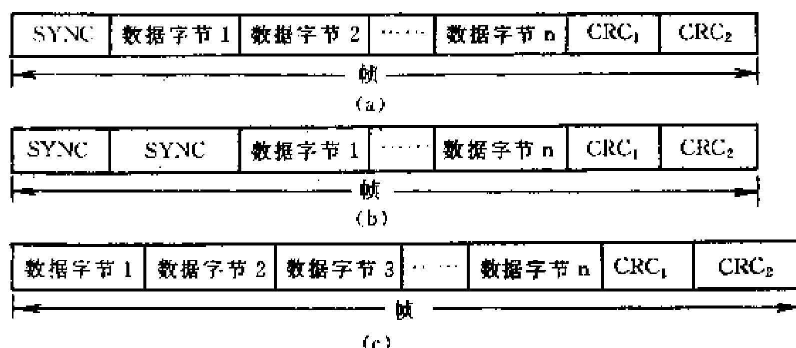
每秒钟可传送数据的 bit 数称为传输率，即波特率(Band Rate)。计算机之间的异步通信速率一经确定后，一般不应变动，但通信的数据是可变动的，也就是数据字之间的空闲位是可变的。

(2) 同步通信协议.

在同步通信时所使用的数据格式根据控制规程分为：面向字符及面向比特的两种。

①．面向字符型的数据式

面向字符型的同步通信数据格式可采用单同步，双同步及外同步三种数据格式，如图 9-14 所示。



(a)单同步　(b)双同步　(c)外同步

图 9-14　面向字符型同步通信数据格式

单同步是指在传送数据之前先传送一个同步字符“SYNC”，双同步则先传送两个同步字符“SYNC”。接收端检测到该同步字符，使接收方与发送端实现同步。当每一帧信息结束时均用两个字节的循环校验码 CRC 结束。

② 面向比特型的数据格式

根据同步数据链路控制规程(SDLC)，面向比特型的数据以帧为单位传输，每帧由六个部分组成。第一部分是开始标志“7EH”；第二部分是一个字节的地址场；第三部分是一个字节的控制场；第四部分是需要传送的数据，数据都是位(bit)的集合；第五部分是两个字节的循环校验码 CRC；最后部分又是“7EH”，作为结束标志。面向比特型的数据格式见图 9-15。

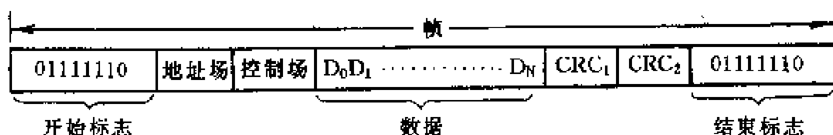


图 9-15　面向比特型的数据格式

在 SDLC 规程中不允许在数据段和 CRC 段中出现六个“1”，否则会误认为是结束标志。因此要求在发送端进行检验，当连续出现五个“1”，则立即插入一个“0”，到接收端要将这个插入的“0”去掉，恢复原来的数据，保证通信的正常进行。

通常，异步通信速率要比同步通信的低。

最高同步通信速率可达到每秒 800K 位，因此适用于传送信息量大，要求传送速率很高的系统中。

3. 串行通信的物理标准

计算机和计算机之间的通信问题，涉及短距离通信，也涉及长距离通信，而且，通信形式也可能是各种各样的，为了使通信双方相互衔接，也为了使计算机及其它通信设备互相沟通。必须对串行通信建立一致的概念和标准，它们包括：传输率、电特性、信号名称和接口标准。

(1)传输率

所谓传输率就是指每秒传输多少位，即波特率。国际上规定了一个标准波特率系列，标准波特率也是最常用的波特率，标准波特率系列为110、300、600、1200、1800、2400、4800、9600和19200。

大多数接口的接收波特率和发送波特率可以分别设置，而且，可以通过编程来指定。作为例子，我们可以考虑这样一个异步传输过程：设每个字符对应1个起始位，7个信息位，1个奇/偶校验位和1个停止位，如果波特率为1200，那么，每秒钟能传输的最大字符数为1200/10=120个。

作为比较，我们再来看一个同步传输的例子。假如也用1200的波特率工作，用4个同步字符作为信息帧头部，但不用奇/偶校验，那么，传输100个字符所用的时间为7(100+4)/1200=0.6067s，这就是说，每秒钟能传输的字符数可达到100/0.6067=165个。可见，在同样的传输率下，同步传输的实际字符传输率要比异步传输时高。

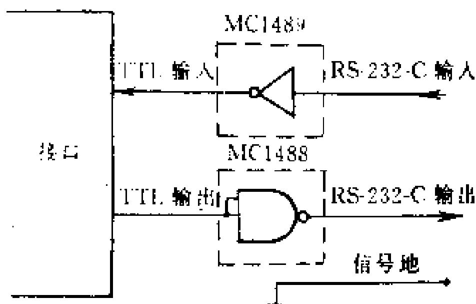


图 9-16　TTL 和 RS-233C 之间的电平转换

(2)RS-232C 标准

RS-232C 标准对下述两方面作了规定：

■ 信号电平标准；

■ 控制信号的定义。

RS-232C 采用负逻辑电平，信号电平与通常的TTL电平也不兼容。RS-232C 将-5V～-15V 规定为“1”，+5V～+15V 规定为“0”。

图9-16是TTL标准和RS-232C标准之间的电平转换电路。

从图9-16中可以看到，要从TTL电平转换成RS-232C电平时，中间要用到MC1488器件，反过来，用MC1489器件，则将RS-232C电平转换成TTL电平。

对接插件引线上控制信号进行定义也是RS-232C标准的一部分，表9-1列出了引线和信号之间的对应关系。RS-232C规定用25芯插头座进行连接，表中第一列是25芯接插件的引脚号，第二列数字是CCITT标准对应的引脚号，后面两栏分别是信号名称和简要说明。

表 9-1　RS-232C 的信号定义

引脚号	CCITT	名　称	说　明
1	101	保护地	作为设备接地端
2	103	发送数据(出)	将数据送调制解调器
3	104	接收数据(入)	从调制解调器接收数据
4	105	请求发送	在半双工方式下控制发送器的开或关
5	106	允许发送	指出调制解调器准备好发送
6	107	数据终端准备好	指出调制解调器不处在测试模式，而是可进入工作状态
7	102	信号地	作为所有信号的公用地
8	109	载波信号检测	指出调制解调器正在接收另一端送来的信号
9		空	
10		空	
11		空	

（续）

引脚号	CCITT	名　称	说　明
12		第二通道接收信号检测	指出在第二通道上检测到信号
13		第二通道允许发送	指出在第二通道准备好发送
14	118	第二通道发送数据	往调制解调器以较低速率输出
15	113	发送器定时	为调制解调器提供发送器定时信号
16	119	第二通道接收数据	从调制解调器以较低速率输入
17	115	接收器定时	为接口和终端接收器提供信号定时
18		空	
19		第二通道请求发送	闭合第二通道的发送器
20	108	数据终端准备好	调制解调器连接到链路，并开始发送
21		空	
22	125	音响指示	指出在链路上检测到音响信号
23	111	数据率选择	可选择两个同步数据率之一
24	114	发送器定时	为接口和终端提供发送器定时信号
25		空	

二、可编程串行通信接口芯片 8251A

（一）8251A 的内部结构

8251A 的内部结构如图 9-17 所示

整个 8251A 有五个主要组成部分：接收器、发送器、调制解调控制逻辑、读写控制逻辑和数据总线缓冲器。

1．接收器

接收器包括接收缓冲器和接收控制逻辑两部分。

（1）接收缓冲器

接收缓冲器主要由移位寄存器和数码寄存器组成。接收器接收传送到 RxD（接收数据输入端）引脚上的串行数据，并对串行数据流的特殊位（奇偶位，停止位等）和字符（同步字符）进行检查、处理，按规定的格式将串行数据转换为并行数据存放在缓冲器中。

接收移位寄存器和接收数据缓冲器组成了双缓冲器结构。

（2）接收控制逻辑

这一部分控制串行数据的接收，包括三条控制线：

RxRDY（Receiver Ready）接收器准备好，输出，高电平有效。

RxC（Receiver Clock）　接收时钟，输入。

SYNDET/BRKDET（SYNchronous DETect/BReaK DETect）同步检测/断点检测，输出/输入，高电平有效。

2．发送器

发送器包括发送缓冲器和发送控制逻辑两部分。

（1）发送缓冲器和发送过程

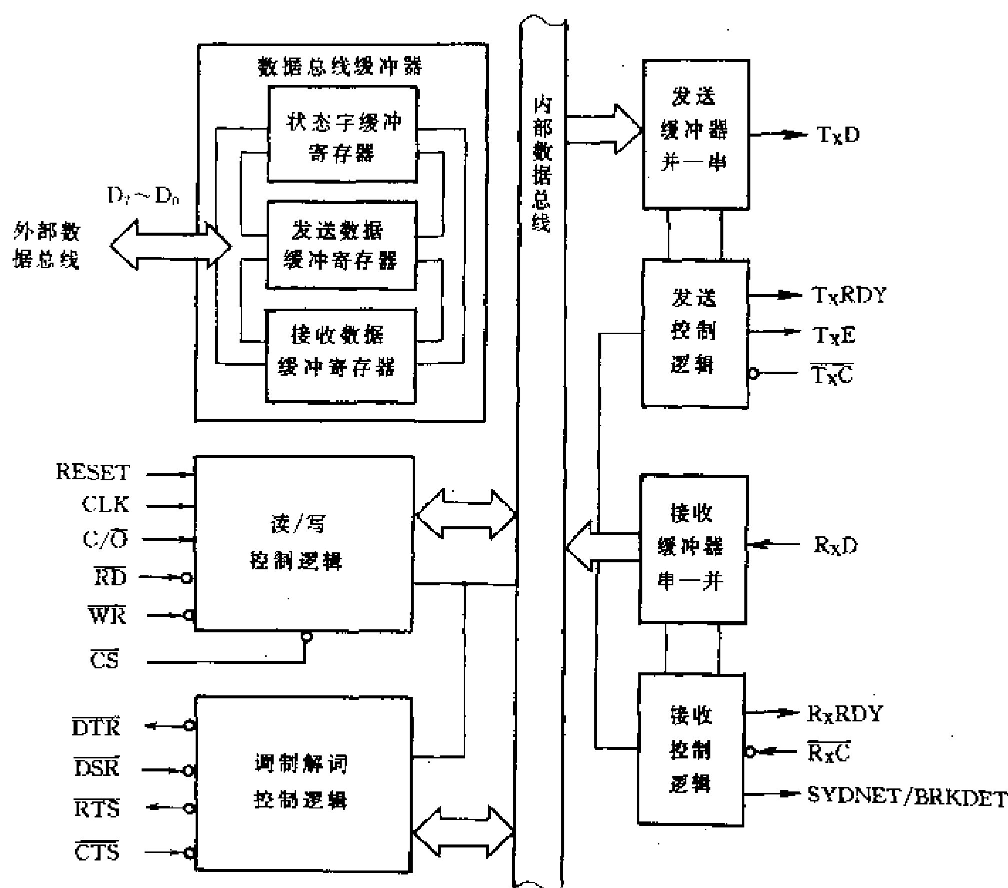


图 9-17　8251A 的内部结构图

发送数据缓冲器接收由 CPU 送来的并行数据，按初始化编程指定的数据格式转换成串行数据流送至发送移位寄存器，在 TxC 的下降沿从 TxD 引脚发送出去。

发送数据缓冲器和发送移位寄存器组成了发送的双缓冲器结构。

(2)发送控制逻辑

该部分控制串行数据的发送操作，包括两条控制线：

TxRDY(Transmitter ReaDY)　发送器准备好，输出，高电平有效。

TxE(Transmitter Empty)　发送器空，输出，高电平有效。

TxC(Transmitter Clock)　发送时钟，输入。

3. 读/写控制逻辑

读/写控制逻辑接收 CPU 的有关控制信号，据此确定对 8251A 的操作。该部分共有 6 条对外引线。

CLK(Clock)　时钟，输入。

RESET　复位，输入，高电平有效。RESET 有效，8251A 被强迫复位到空闲状态。只有在重新初始化后才能脱离空闲状态。

$\overline{CS}$(Chip Select)　片选，输入，低电平有效。

C/$\overline{D}$(Control/Data)　控制/数据信号，输入。

$\overline{RD}$(Read)　读，输入，低电平有效。

$\overline{WR}$(WRite)　写，输入，低电平有效。

4. 数据总线 I/O 缓冲

数据总线缓冲器以三态双向的形式经引脚 $D_7 \sim D_0$ 和系统的数据总线相连。数据总线缓冲器包括：

(1) 状态字缓冲寄存器，寄存 8251A 接收/发送操作的各种工作状态。

(2) 发送数据缓冲寄存器，暂存由 CPU 送来的数据或控制字。8251A 没有独立的控制寄存器，写入的控制命令和发送的数据共用一个寄存器。

(3) 接收数据缓冲寄存器，暂存接收到的准备送往 CPU 的数据。

5. 调制解调控制逻辑

远程通信时，8251A 的 TxD 端数据经调制器调制后送上传输线，经传输线送来的信号经解调后送往 8251A 的 RxD 端。为了在 8251A 和调制解调器之间能正确地传送数据，8251A 调制解调控制逻辑产生四个相应的联络信号如下：

$\overline{DTR}$(Data Terminal Ready)	数据终端准备好，输出，低电平有效。
$\overline{DSR}$(Data Set Ready)	调制解调器准备好，输入，低电平有效。
$\overline{RTS}$(Request To Send)	请求发送，输出，低电平有效。
$\overline{CTS}$(Clear To Send)	允许发送，输入，低电平有效。

当 8251A 不与调制解调器相接而是接续其它外设时，这四条线可以作为控制数据传输的联络线。

(二) 8251A 芯片的引脚及其功能

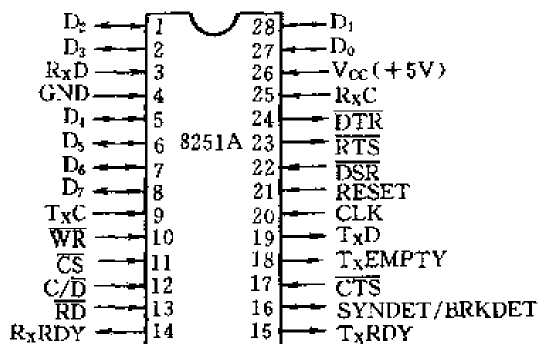


图 9-18　8251A 引脚图

如图 9-18 为 8251A 的芯片引脚示意图。

作为 CPU 和外部设备(或调制解调器)之间的接口，8251A 的对外信号分为两组，一组是 8251A 和 CPU 之间的信号，一组是 8251A 和外部设备(或调制解调器)之间的信号。

1. 8251A 和 CPU 之间的连接信号

8251A 和 CPU 之间的连接信号可分为 4 类，具体如下：

(1) 片选信号

$\overline{CS}$　片选信号$\overline{CS}$是 CPU 的地址信号通过译码后得到的。$\overline{CS}$为低电平时，8251A 被选中。反之，$\overline{CS}$为高电平时，8251A 未被选中，这种情况下，8251A 的数据线处于高阻状态，读信号$\overline{RD}$和写信号$\overline{WR}$对芯片不起作用。

(2)数据信号

$D_7 \sim D_0$　8251A 有 8 根数据线 $D_7 \sim D_0$，通过它们，8251A 与系统的数据总线相连。实际上，数据线上不只传输一般的数据，而且也传输 CPU 对 8251A 的编程命令和 8251A 送往 CPU 的状态信息。

(3)读/写控制信号

$\overline{RD}$　当读信号$\overline{RD}$为低电平时，用来通知 8251A，CPU 当前正在从 8251A 读取数据或者状态信息。

$\overline{WR}$　当写信号$\overline{WR}$为低电平时，用来通知 8251A，CPU 当前正在往 8251A 写入数据或者控制信息。

$C/\overline{D}$控制/数据端口选择信号，用来区分当前读写的是数据还是控制信息或状态信息。具体地说，CPU 在读操作时，如 $C/\overline{D}$为低电平，则读取的是数据，如 $C/\overline{D}$为高电平，则读取的是 8251A 当前的状态信息；CPU 在写操作时，如 C/D 为低电平，则写入的是数据，如 C/D 为高电平，则写入的是 CPU 对 8251A 的控制命令。

归纳起来，$\overline{RD}$、$\overline{WR}$、$C/\overline{D}$这 3 个信号和具体的读/写操作之间的关系如表 9-2 所示。

表 9-2 $\overline{RD}$、$\overline{WR}$、$C/\overline{D}$的编码和对应的操作

$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	具体的操作
0	0	1	CPU 从 8251A 输入数据
0	1	0	CPU 往 8251A 输出数据
1	0	1	CPU 读取 8251A 的状态
1	1	0	CPU 往 8251A 写入控制命令

（三）8251A 的编程：

8251A 是一个可编程的多功能通信接口电路，在用它传送数据之前必须对它进行初始化编程，确定它的具体工作方式。改变 8251A 的工作方式，也必须要对其再次进行初始化编程。

1. 控制字

(1) 工作方式控制字

工作方式控制字在复位后写入，它详细规定了 8251A 的工作方式，其格式如图 9-19 所示。

D_1D_0　确定是工作于同步方式还是异步方式。$D_1D_0=00$ 为同步方式；$D_1D_0\neq 00$ 为异步方式，且有 3 种组合来选择输入的时钟频率与波特率之间的系数。接收和发送的波特率系数只能是同一个，接收和发送的波特率相同与否视 RxC 和 TxC 的频率是否相同。

D_3D_2　确定每个字符的数据位(不包括奇偶校验位)。当字符长度少于 8 位时有效数据位是右对齐的，数据写入 8251A 时没用到的位可以随意设置，接收数据时没有用到的位都是 0。

D_5D_4　确定奇偶校验。

D_7D_6　含义因同步方式而异。异步方式($D_1D_0\neq 00$)时用来确定停止位个数。同步方式时 D_6 用来确定是内同步(SYNDET 脚为输出)还是外同步(SYNDET 为输入)，D_7 用来确定同步字符个数。外同步方式时，同步字符只用于发送，接收时不起作用。

(2) 同步字符

当方式设为同步($D_1D_0=00$)时，方式控制字后必须装入同步字符，并由同一个方式控制字规定装入单同步字符($D_7=1$)还是双同步字符($D_7=0$)。

(3) 命令控制字

同步方式在同步字符之后(或异步方式在方式字之后)接着写入命令控制字。写入方式字只规定了 8251A 的工作方式，并不能使 8251A 开始工作。只有写入命令字后才能使 8251A 处于相应的运行状态，接收或发送数据。异步方式在方式字之后(或同步方式在同步字符之后)所写入的控制信息($C/\overline{D}=1$)只能是命令控制字。只有在复位(外部复位或由命令字复位)之后才能使 8251A 返回到写方式字状态，才能写入方式字改变 8251A 的工作方式。命令字的格

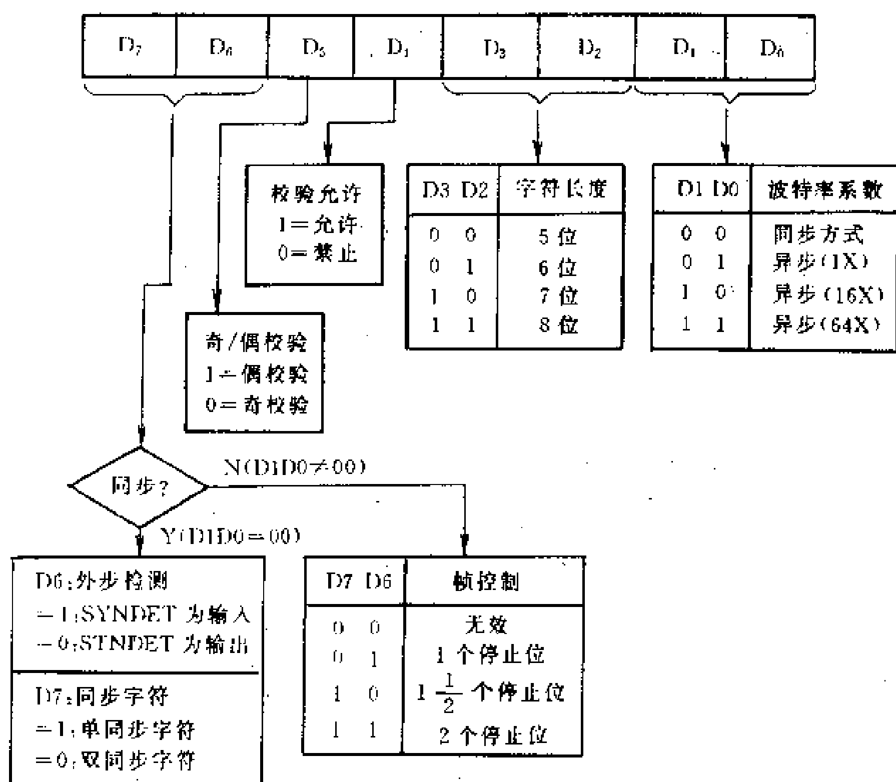


图 9-19 8251A 工作方式控制字格式

式如图 9-20 所示。

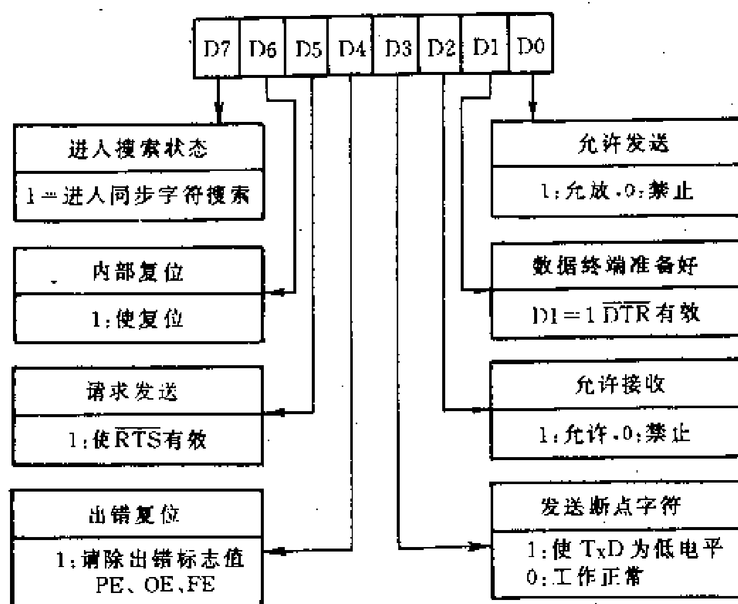


图 9-20 8251A 命令字格式

D_0 设置为 1 允许 8251A 开始发送操作。只有命令字的 $D_0=1$,引脚 TxDRY(通知 CPU:发送器准备好)才可能有效(为 1)。

D_1 设置为 1 强制引脚 DTR(数据终端准备好)有效,通知调制解调器:8251A 已准备好。

D_2　设置为 1 允许 8251A 开始接收数据。只有命令字 $D_2=1$，RxRDY(通知 CPU 接收器准备好的引脚)才有可能为 1。允许接收时必须使错误标志复位(见 D_4)。在同步方式时还必须指定进入同步搜索操作(见 D_7)。

D_3　设置为 1 迫使 TxD 端发送低电平，以此作为断点字符。

D_4　设置为 1 则对状态字中的所有操作出错标志(FE，OE，PE)复位。

D_5　设置为 1 强制 RTS 引脚(请求发送)有效，向调制解调器提出发送请求。

D_6　设置为 1 强制 8251A 内部复位，使之回到准备接收方式字的状态。

D_7　只用于同步方式。为使 8251A 进入同步搜索操作，将输入的信息和同步字符比较，一致则使 SYNDET/BRKDET(同步/断点检测)引脚有效，开始对数据的接收操作。

2. 状态字。

8251A 的状态字格式如图 9-21 所示。

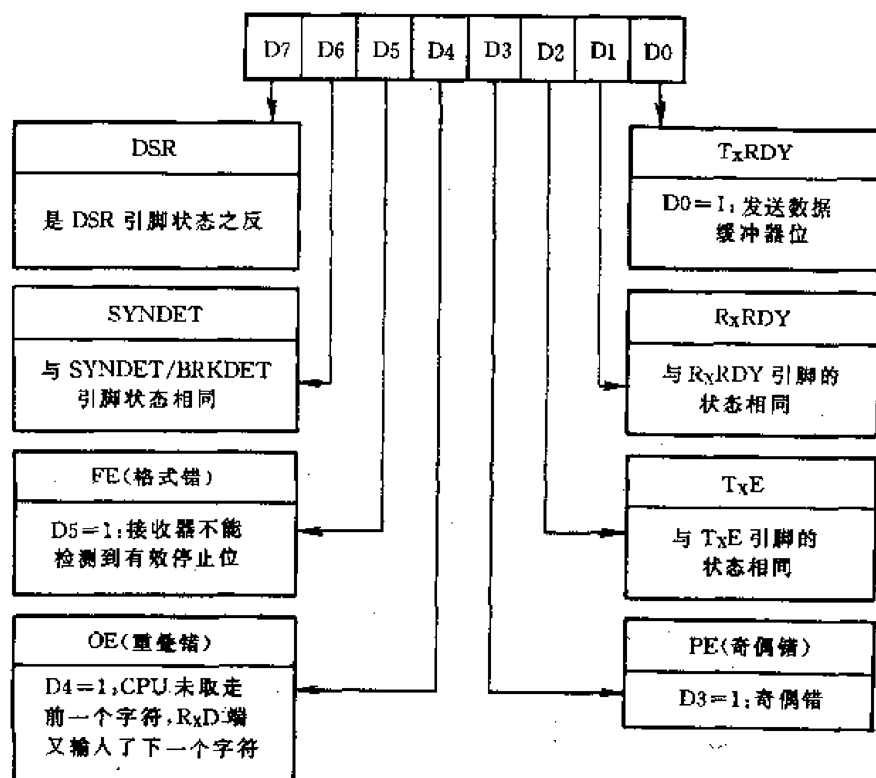


图 9-21　8251A 状态字格式

方式字，同步字符，命令字都是 CPU 写入 8251A 的，以控制 8251A 的工作方式和操作。那么，8251A 在发送，接收数据的过程中实际工作状态如何呢？如一个字符接收全了没有？接收的数据有没有错误？有什么类型的错误？发送缓冲器空了没有？发送移位寄存器空了没有？等等，这些在发送/接收数据操作过程中的状态信息随时寄存在 8251A 内部的状态缓冲寄存器内，CPU 可以通过 I/O 读操作(C/D=1)把状态字读入加以分析，控制 CPU 和 8251A 之间的数据交换。

状态字的 D_1(RxRDY)和 D_0(TxRDY)可供 CPU 查询。

状态位 TxRDY 只反映发送命令/数据缓冲器的实际状态，只要数据缓冲器一空就置 1，而输出引脚 TxRDY 还要受到命令字中允许发送位 TXEN(D_0)和引脚 $\overline{CTS}$(允许发送，输入)的控制，不仅仅反映发送过程中数据缓冲器的状态。它们的条件分别为：

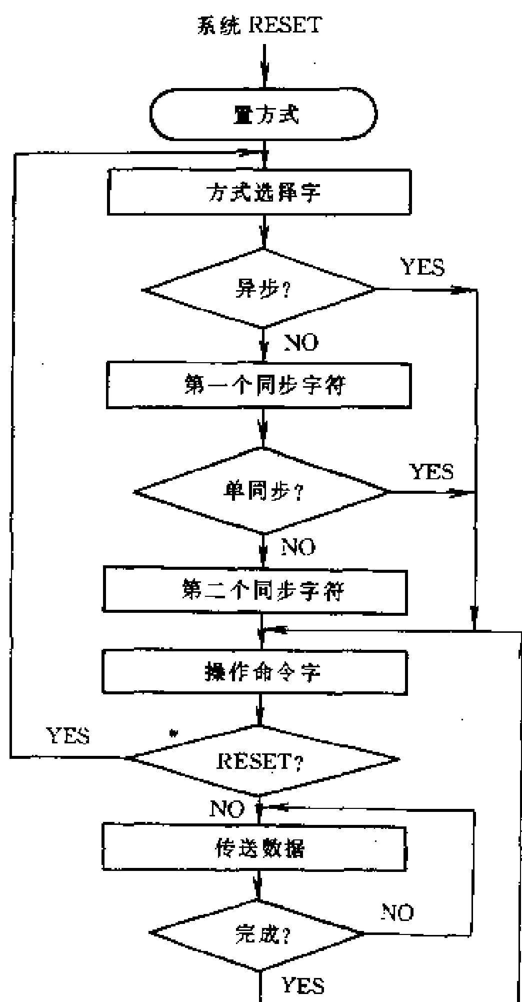


图 9-22　8251A 初始化的编程流程图

状态 TxRDY＝数据冲器空。

输出引脚 TxRDY＝数据缓冲器空·(CTS＝0)·(TxEN＝1)

发送前及发送后两者状态可能不一致，但在发送过程中这两者的状态总是一致的。状态位 TxRDY 可供 CPU 查询，输出引脚 TxRDY 可作为中断请求信号。

CPU 读状态期间，8251A 自动禁止状态位的改变，但最大延迟时间不超过 28 个时钟周期。

3. 初始化编程

8251A 只能根据写入的顺序来识别控制字的种类。初始化编程先写入方式字，但方式字必须在 8251A 复位后写入。8251A 复位有两种方法：系统复位，打开电源或按下系统复位键后所产生的复位脉冲将通过 RESET 端使 8251A 复位；内部复位，以编程方法写入 $D_6=1$ 的命令字使之复位。

初始化编程写入控制字的顺序如图 9-22 所示。

例如，8251A 工作于异步方式，方式选择控制字为 11111011B，操作命令控制字为 00010001B。其初始化程序为：

```
      ⋮
MOV   AL, 0FBH      ;8251A 方式选择字
OUT   CONTR, AL
MOV   AL, 11H       ;8251A 操作命令字
OUT   CONTR, AL
      ⋮
```

（四）8251A 的应用举例

在微机系统中，多使用异步通信方式，下面以微机系统中 CRT 显示器-键盘串行通信接口为例，说明 8251A 的应用

1. 硬件连接图

硬件连接电路如图 9-23 所示。

8251A 主时钟 CLK 为 2MHz，发送时钟 TxC 和接收时钟 RxC 由 8253 的计数器 2 输出提供。片选信号 CS 由地址线高位译码后提供。数据线 $D_7 \sim D_0$ 与该系统的 16 位数据总线的低 8 位 $D_7 \sim D_0$ 相连。

8251A 经 RS-232C 接口与显示器、键盘相连，所以 8251A 和显示器、键盘的 RS-232C 接

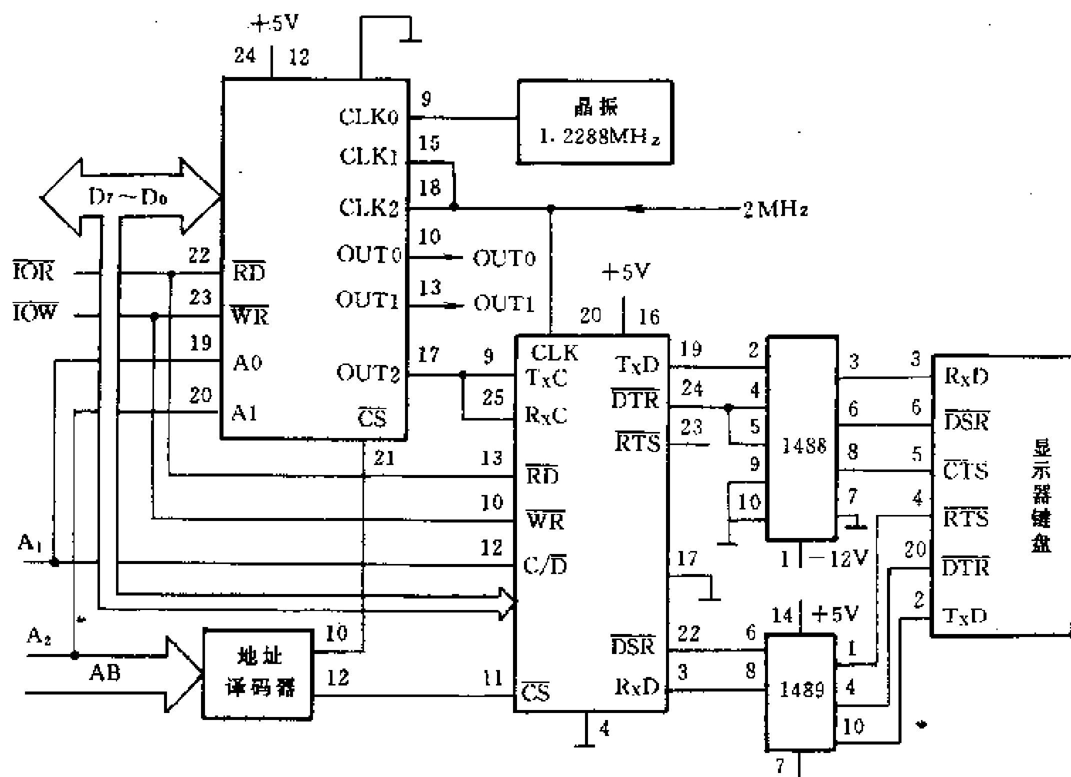


图 9-23 8251A 用作 CRT 接口的电路连接

口间要用 1488 和 1489 进行电平变换。

2. 8251A 初始化程序

```
                                  ;8251A 初始化编程
INITIA: MOV     AX, 0
        MOV     CX, 0003H
        MOV     DX, 00DAH
OTPT:   CALL    XYZ
        LOOP    OTPT
        MOV     AL, 40H
        CALL    XYZ
        MOV     AL, 4EH        ;设置 8251A 为异步方式,波特率因子为 16
        CALL    XYZ            ;8 位数据位,一位停止位
        MOV     AL, 27H        ;命令 8251A 发送器和接收器启动
        CALL    XYZ
           ⋮
;输出子程序,将 AL 中数据输出到 DX 指示的端口
XYZ:    OUT     DX, AL
        PUSH    CX
        MOV     CX, 0002H
YD:     LOOP    YD              ;等待输出动作完成
```

```
        POP     CX
        RET
```

3. 发送子程序

将输出到 CRT 的字符事先放在堆栈中,发送程序先对状态字的 TxRDY 位进行测试;若为 1 表示发送缓冲器已空,CPU 向 8251A 输出一个字符供其继续向 CRT 发送。

```
;发送子程序
 SOUT:  MOV    DX, 0DAH
SOUSR:  IN     AL, DX
        TEST   AL, 01H
        JZ     SOUSR
        MOV    DX, 0D8H
        POP    AX
        OUT    DX, AL
        RET
```

4. 接收子程序

从键盘接收的字符放入 AL 中,接收程序先对状态字的 RxRDY 位进行测试,若为 0 表示接收的数据未准备好,暂时不能输入数据,继续测试等待;若为 1 表示接收缓冲器中来自键盘的数据已准备好,CPU 可从 8251A 输入一个字符至 AL 中,然后再按要求进一步处理。

```
;接收子程序
  SIN:  MOV    DX, 0DAH
 SIST:  IN     AL, DX
        TEST   AL, 02H
        JZ     SIST
        MOV    DX, 0D8H
        IN     AL, DX
         ⋮
```

第三节　计数/定时控制器接口技术

一、概述

在计算机系统中经常要用到定时信号。例如，在许多微型计算机中动态存储器的刷新定时,系统日历时钟的计时以及喇叭的声源,都是用定时信号来产生的. 又比如在计算机实时控制与处理系统中计算机主机需要每隔一定的时间就对处理对象进行采样，再对获得的数据进行处理,也要用到定时信号。

一般来说,定时信号可以用软件和硬件两种方法获得。

本节主要讲述计数/定时器的基本原理,介绍 8086/8088 系统中广泛采用的计数/定时器芯片 INTEL 8253,最后给出一个 8253 的应用举例。

二、可编程计数/定时器芯片 8253

(一) 8253 的内部结构

8253 的内部结构如图 9-24 所示,由 CPU 的接口部分、控制逻辑和三个计数器组成。

(1) 数据总线缓冲器

这是 8253 连接 CPU 数据总线的 8 位双向三态缓冲器。CPU 读/写 8253 的所有数据都经过这个缓冲器。

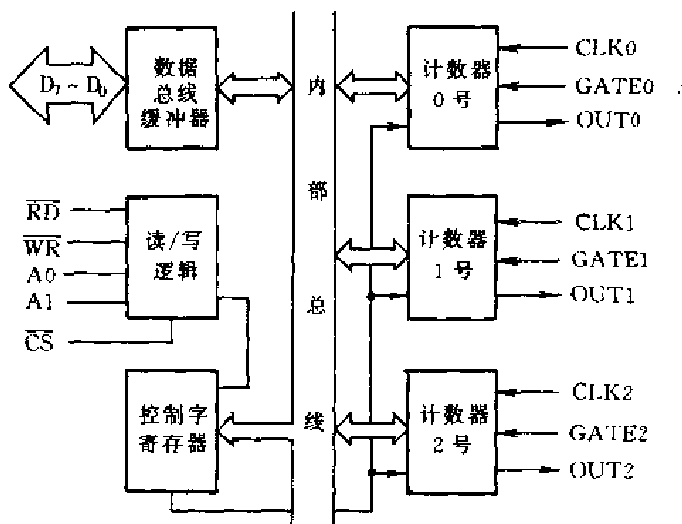


图 9-24　8253 内部结构图

CPU 用输出指令向 8253 写入方式控制字至控制寄存器,或写入计数值至某个计数器,都是经数据总线缓冲器和 8253 内部总线传送的。CPU 用输入指令读某个计数器时,指定计数器的现行计数值经内部总线和缓冲器传送到系统数据总线上,读入 CPU。

(2) 读/写逻辑

这是 8253 内部操作的控制部件,接收系统总线的输入信号,转换成 8253 内部操作的各种控制信号,选择读/写操作的对象(计数器或者控制寄存器),决定内部总线上数据传送的方向。 2

读/写逻辑接受$\overline{CS}$芯片选择信号控制。$\overline{CS}$为 1 时读/写被禁止,CPU 不能对 8253 进行读/写,数据总线缓冲器呈高阻状态,与系统的数据总线脱离。这时对芯片设置的工作方式和各计数器的现行工作不受影响。当$\overline{CS}$为 0 时 CPU 可读/写计数器或写控制字到控制寄存器中。

(3) 控制字寄存器

当$\overline{CS}=0$ 且 $A_1A_0=11$ 时,该寄存器接收 CPU 写入 8253 的控制字。控制字控制指定计数器的工作方式,选择二进制或二一十进制计数,规定读/写的具体方法。

控制寄存器只能写入,其内容不能读出。

(4) 计数器 0、计数器 1、计数器 2

每个计数器是一个 16 位减一计数器,可接收计数值寄存器预置的初值,三个计数器的内部结构是相同的,它们的操作各自完全独立。

每个计数器都可对其 CLK 输入端输入的脉冲按照二进制或二一十进制从预置初值开始减 1 计数。当计数器减到 0 时从 OUT 输出端输出一个脉冲信号。当 CLK 端输入周期恒定的脉冲时,计数器就完成了定时功能。

计数的开始和整个计数过程,计数器可以受 GATE 输入端的门控信号控制。与外界相连的计数器输入输出以及门控信号之间的关系取决于该计数器的工作方式。

计数器的初值必须在计数之前由 CPU 用输出指令预置,在计数过程中,CPU 能随时用输入指令读入计数器的当前计数值,而不影响计数器的计数。

(二) 8253 芯片的引脚及其功能

8253 是双列直插式 24 脚封装，引脚如图 9-25。

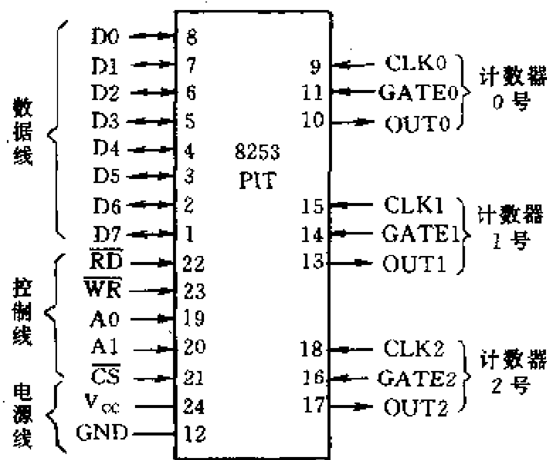


图 9-25 8253 引脚图

8253 的三个计数器功能是完全一样的，在整个结构上不仅内部逻辑而且外部引脚都是相同的。每个计数器有三个引脚：脉冲输入、计数输出和门控输入。

CLK 脉冲输入引脚。

计数器对该引脚输入的脉冲进行计数。8253 的基本工作方式是对 CLK 端输入的脉冲进行计数。CLK 输入脉冲可以是任何脉冲源提供的脉冲(只要它的周期不小于 380ns 即可)，包括系统时钟和系统时钟分频后的脉冲。

GATE 门控脉冲输入引脚。

这是外部控制计数器工作的门控脉冲输入引脚。通常，GATE 为 0 时禁止计数器工作，GATE 为 1 时允许计数器工作。GATE 也可启动或中止计数器/定时器运行。

OUT 计数到 0/定时时间到输出引脚。

不管工作于何种方式，当计数到 0 时，OUT 引脚定有输出，但在不同的工作方式下输出不同形式的信号。

一片 8253 占四个端口，对应片内的三个计数器和一个控制字寄存器。端口的选择由 8253 的 A_1 和 A_0 输入引脚的状态决定，它们通常接系统地址总线的 A_1 和 A_0(8 位微处理机)或 A_2 和 A_1(16 位微处理机)。片选 $\overline{CS}$、读 $\overline{RD}$、写 $\overline{WR}$ 和 A_1A_0 对 8253 各端口的选择和读/写操作见表 9-3。

表 9-3 8253 端口选择和读/写操作

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	端口选择和操作
0	1	0	0	0	写入计数器 0
0	1	0	0	1	写入计数器 1
0	1	0	1	0	写入计数器 2
0	1	0	1	1	写方式控制字到控制字寄存器
0	0	1	0	0	读计数器 0
0	0	1	0	1	读计数器 1
0	0	1	1	0	读计数器 2
0	0	1	1	1	无操作，数据总线缓冲器三态
0	1	1	X	X	无操作，数据总线缓冲器三态
1	X	X	X	X	禁止，数据总线缓冲器三态

(三) 8253 的工作方式

8253 每个计数器有 6 种工作方式，由控制字的 $D_3D_2D_1$ 确定。在不同的工作方式，计数器完成不同的功能。

(1) 方式 0　计数结束产生中断

8253 用作计数器时一般 工作在方式 0。当控制字写进控制字寄存器确定了方式 0 时，计数器的输出立即变低，即使计数还没有赋初值，也没有开始计数。计数初值装入该计数器之后，在门控 GATE 信号为高电平时计数器开始减 1 计数。当计数器减到 0 时输出端 OUT 才由低变高，此高电平输出一直保持到该计数器装入新的计数值或再次写入方式 0 控制字为止。计数到 0 的输出信号可作为中断请求信号发往 CPU 申请中断。

(2) 方式 1　可编程的单拍负脉冲

CPU 写入控制字后计数器输出为 1，在写入计数值后计数器并不开始计数，而要由外部门控 GATE 脉冲上升沿启动。GATE 上升沿之后的下一个 CLK 输入脉冲的下降沿开始计数，输出 OUT 变低，开始单拍脉冲；直到计数减为 0 输出变高，单拍脉冲结束。若计数初值设为 N，则单拍脉冲宽度为 N 个 CLK 时钟脉冲周期。

(3) 方式 2　分频脉冲发生器

这种方式是输出对输入脉冲按计数器计数值 N 分频后的连续脉冲。当 CPU 写入控制字后输出端为高，在写入计数值 N 后将立即自动开始对输入脉冲 CLK 计数，输出端仍一直为高；当计数器减到 1 时，输出变低，计数器减到 0 时输出又变为高，计数器重新按已写入的计数值 N 继续计数，周而复始，即输出连续脉冲，周期是 N 个 CLK 时钟脉冲周期之和，低电平输出宽度(持续时间) 是一个 CLK 脉冲周期。

(4) 方式 3　分频方波发生器

方式 3 常用于波特率发生器。方式 3 和方式 2 类似。但输出为方波或近似方波的矩形波。写入方式 3 控制字后输出为高。写入计数值后计数器自动开始对输入 CLK 脉冲计数，输出 OUT 仍保持为高；在计数完成一半时，输出 OUT 变为低，直到计数器全部完成输出 OUT 又变为高，并重复上述计数过程。若 N 为偶数时，OUT 方波的占空比为 1:1；若 N 为奇数，其占空比为：(N+1)/2:(N-1)/2。

(5) 方式 4　软件触发选通脉冲发生器

方式 4 是靠写入计数值来进行软件触发的“一次性有效” 的选通脉冲发生器写入控制字后输出端 OUT 变为高，这是起始电平。在写入计数值之后开始计数，当计数到 0 时输出端 OUT 变为低，维持一个 CLK 周期后又恢复为高，并一直保持为高，直到再次写入计数值来进行“软件触发”才能再次开始。方式 4 的负脉冲输出常作为选通脉冲。

(6) 方式 5　硬件触发选通脉冲发生器

方式 5 是靠门控脉冲 GATE 的上升沿来进行触发的选通脉冲发生器。写入控制字后输出端 OUT 为高，这是初始电平；写入计数值后计数器并不开始计数，而要由门控脉冲 GATE 上升沿触发后才开始计数；计数到 0 输出由高变低，一个 CLK 时钟周期后又恢复为高，并一直保持；直到下次门控脉冲触发再次开始计数。

方式 5 输出波形与方式 4 一样，但是方式 5 是硬件触发。

(四) 8253 的编程

1. 8253 计数器初始化编程

8253 每个计数器的工作方式和计数值都必须由 CPU 通过输出指令来设定。8253 计数器初始化包括向该计数器写方式控制字和计数值这两个步骤。

对 8253 初始化的要求是：

(1) 对每个计数器，控制字必须写在计数值之前。这是因为计数器的读/写格式由它的控

制字决定。

(2) 计数值必须按控制字所规定的格式写入。若控制字规定只写 8 位只需写入一次(8 位)计数值即可(规定写低 8 位则高 8 位自动置 0,规定写高 8 位则低 8 位自动置 0),规定写 16 位时必须写两次,先写低 8 位,后写高 8 位。

(3) 对所有方式计数器都可以在计数过程中或计数结束后改变计数值，重写计数值也必须遵守控制字所规定的格式,并且不会改变当前计数器的工作方式。

计数值不能直接写到减 1 计数器中,而只能写入计数值寄存器中,并由写操作 WR 之后的下一个 CLK 脉冲将计数值寄存器的内容装入减 1 计数器开始计数。

初始化编程必须明确各个计数器的控制字和计数值不是写到同一个地址单元。各个计数器的控制字各自独立确定,但它们都写入同一个端口地址(控制寄存器)中,各个计数器的计数值则根据需要独立确定并写入各自计数器的相应寄存器中。

2. 8253 读命令

读命令只能用于读取计数器的当前值,控制字是不能读出的 8253 内部逻辑允许在不干扰实际计数过程的情况下很容易地读出计数器的值。读操作必须按控制字规定的格式，从每个计数器对应的端口地址中读出;如果是 16 位计数(计数值写入两字节),读计数值须进行两次,先读低 8 位,后读高 8 位。

读 8253 计数器的计数值有两种方法:直接读出和锁存读出。

(1) 直接读出

直接用输入指令读取所选择的端口计数器。因为计数器当前值要分两次读入，而每次读入后总要将读入的值存放到某个寄存器或存储单元之中，因此两次读数之间必定有段时间间隔,而经过这段时间计数值可能已经变化了,这就造成读数的不准确。所以直接读计数值的条件是在读数之前必须先用门控脉冲或外部逻辑禁止所读计数器计数,停止计数后再读。这要求内部和外部、软件和硬件相互配合,给使用带来一定麻烦,不常用。

(2) 锁存读出

锁存计数值以供读取,是专为在计数过程中读数据而设计的。锁存命令是控制字的一个特殊形式:$D_5D_4=00$(锁存),$D_7D_6=$计数器号,低 4 位可设为全 0。读数之前先执行一次写操作,将所要读的计数值锁存,然后再来两次读操作,读出锁存命令写入时刻计数器的计数值。

锁存命令并不妨碍计数器正常计数。每个计数器都有两个 8 位输出锁存器,分别锁存计数器的高 8 位和低 8 位。在计数器计数过程中,没来锁存命令时,输出锁存器随计数器的变化而变化;当接收到计数器锁存命令时,输出锁存器中的计数值就被冻结,而计数器仍正常计数,随输入 CLK 脉冲而变化。锁存器不再跟随变化,直到锁存器中数据被读走或者计数器重新编程,输出锁存器随之自动解除锁存状态而又随着计数器变化 。

锁存命令后不管什么时候读,读到的都是锁存命令写入时刻的计数值。再次读数,应再次写入锁存命令,然后再读数。这种方法比直接读出多用两条指令(输出锁存命令),但却省去了对外部硬件的要求。如果读数之前使用了多次锁存命令，则第一次以后的锁存命令可能是无效的,可能读到的是执行第一次锁存命令时的计数值。

(五) 8253 的应用举例

图 9-26 所示是 IBM PC/XT 系统板中 8253-5 的接口电路。它的三个计数器的使用情况如下。

PCLK 是来自时钟发生器 8284A 的输出时钟,频率 2.38MHz,经 74LS175 二分频后作为

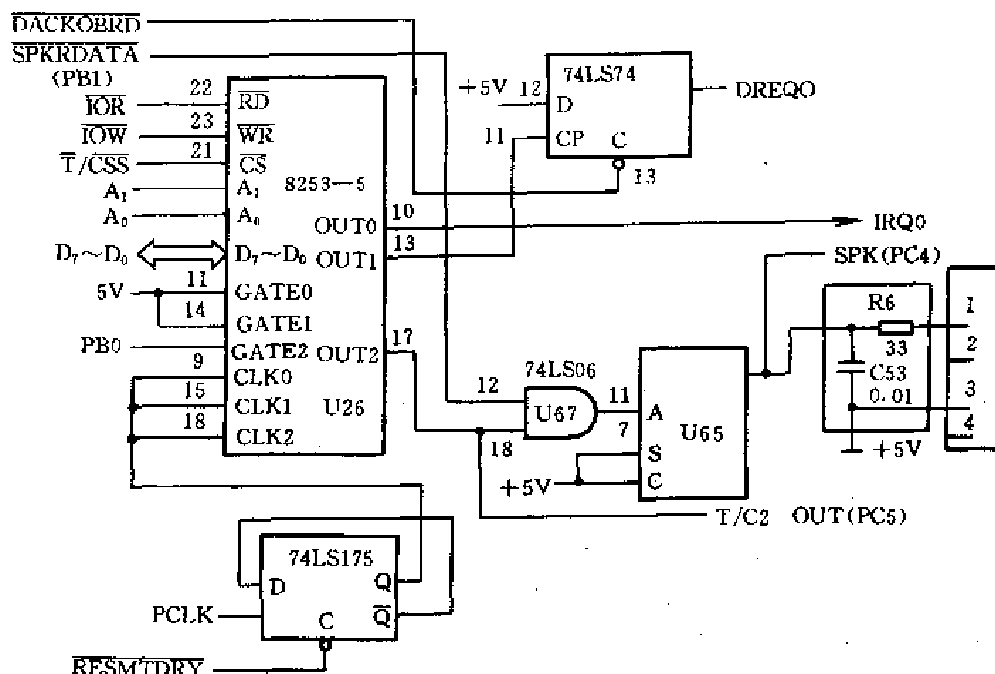


图 9-26 IBM-PC/XT 系统板中 8253-5 的接口电路

8253-5 的三个计数器的脉冲输入。

1. 计数器 0 编程为方式 3，$GATE_0$ 固定为高电平，作为中断请求接至 8259A 中断控制器的 IRQ_0，此定时中断(约 55ms)用于报时时钟和磁盘驱动器的定时。

2. 计数器 1 编程为方式 2，$GATE_1$ 固定为高电平，OUT_1 的输出经 74LS74 后作为对 8237A-5DMA 控制器通道 0 的 DMA 服务请求 $DREQ_0$，用于定时(约为 15μs)启动刷新动态存储器 DRAM。

3. 计数器 2 编程为方式 3，1KHz 的方波输出，滤掉高频分量后送到扬声器，它的门控信号 $GATE_2$ 是 8255-5 的 PB_0，OUT_2 输出亦经一与门 74LS06 控制，控制信号为 8255A 的 PB_1、可用 PB_1，PB_0 同时为高的时间来控制发音时间。长音时间为 3 秒；短音时间为 0.5 秒.

8253-5 的端口地址为 40H～43H。

对应三个计数器的相应的初始化程序如下(这些程序已在 ROM-IOS 中)

(1) 计数器 0 用于定时(约 55ms)中断。

```
MOV     AL, 0011010B      ;16 位二时制计数，方式 3
OUT     43H, AL
MOV     AL, 0             ;初值为 0000，即为最大值
OUT     40H, AL           ;OUT 两次变"高"之间的间隔
OUT     40H, AL           ;为 840ns × 65536＝55ms
```

(2) 计数器 1 用于定时(约 15μs)DMA 请求.

```
MOV     AL,01010100B      ;只装低 8 位计数器，方式 2
OUT     43H, AL
MOV     AL, 12H           ;初值 18，OUT 两次变高之间的间隔为
OUT     41H, AL           ;840ns×18＝15μs，2ms 内可刷新 132 次
```

(3) 计数器 2 用于产生 1KHz 的方波送至扬声器发声，声响子程序为 BEEP，入口地位为

FFA08H。

```
BEEP  PROC   NEAR
      MOV    AL, 10110110B        ;16 位二进制计数器,方式 3
      OUT    43H, AL              ;写入计数器 2 的控制寄存器
      MOV    AX, 0533H            ;初值为 1331
      OUT    42H, AL              ;先写低字节
      MOV    AL, AH               ;再写高字节
      OUT    42H, AL
      IN     AL, 61H              ;读 8255 的 B 口原输出值
      MOV    AH, AL               ;存于 AH
      OR     AL, 03H              ;使 PB1,PB0 均为 1
      OUT    61H, AL              ;输出以使扬声器发声
      SUB    CX, CX               ;CX 为循环计数,最大 65536
GT:   LOOP   GT                   ;循环延时
      DEC    BL                   ;BL 为发声长短的入口条件
      JNZ    GT                   ;BL=6 发长声,BL=1 发短声
      MOV    AL, AH               ;取回 AH 中的 8255A 的 B 口输出值
      OUT    61H, AL              ;恢复 8255 的 B 口,停止发声
      RET                         ;返回
```

第四节　多功能 I/O 接口芯片 82380

随着半导体器件制造业的发展,微型计算机 I/O 接口电路的集成度越来越高，功能越来越强。原来多片单一功能的 I/O 电路，现在已可能做到一只超大规模集成芯片中。本节以 80386/80486 微型计算机系统中广泛使用的 82380 为例,简要说明一下多功能 I/O 接口电路的构成及各部分工作原理。

一、82380 的结构

82380 的功能框图示于图 9-27。从 I/O 功能上看,它包括了 DMA 控制器、中断控制器和定时器。除此之外,它还具有许多系统支援功能,如等待状态控制、DRAM 刷新控制及系统复位控制等。因此它又被称作集成系统外围支援器件(integrated systemsupport peripherals)。

82380 和 80386/80486 的接口非常简单,可以将其对应引线直接连到 CPU 总线上。一般情况下(例如每次系统复位后),82380 工作于从属(slave)状态，作为系统总线上的一个从属器件,接收 CPU 的控制。这时 82380 被当成一个 8 位的器件。当 CPU 对它写入时，写入字节通过数据总线的 $D_7 \sim D_0$ 或 $D_{15} \sim D_8$(由字节使能信号 BE0＃～BE3＃决定)进入 82380 中。当 CPU 对它读出时，读出的字节数据将依次在数据总线 $D_7 \sim D_0$,$D_{15} \sim D_8$,$D_{23} \sim D_{16}$,$D_{31} \sim D_{24}$上重复出现四次。若进入 DMA 过程,82380 将成为主器件而接管系统总线,这时它和 80386/80486 一样,成为 32 位的器件。

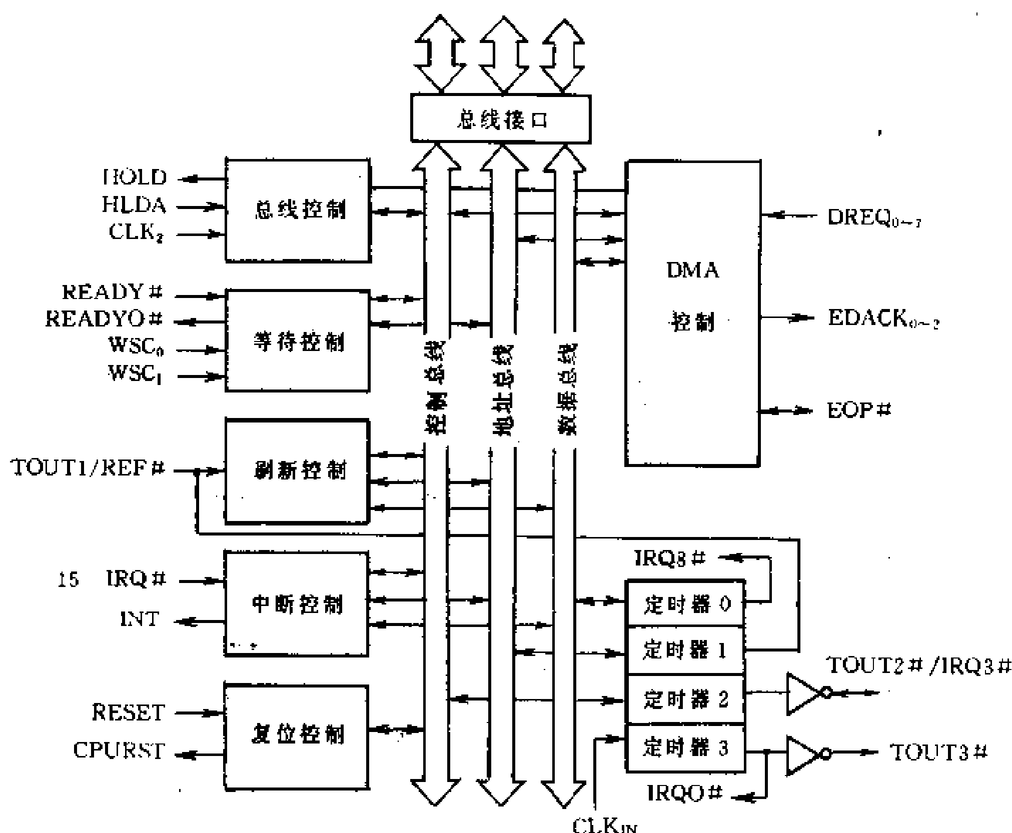


图 9-27 82380 功能框图

二、82380 的 DMA 功能

82380 内部有一个 8 通道的 32 位 DMA 控制器，可以实现内存与内存之间，外部设备与外部设备之间以及内存与外部设备之间的直接传送。传的数据可以是字节、字或双字的任意组合。传送中遇到未对准(misaligned)的字或双字，还可以分解和重新组合。传送的源和目的地址可以加 1、减 1 或保持不变。其地址寄存器为 32 位，最高能覆盖 4GB 的实地址空间。其字节计数器有 24 位，最多可连续传送 16MB。

现结合有关控制信号说明一下 DMA 的工作原理。

$DREQ_7$~$DREQ_0$ 外部 DMA 请求的输入信号。当这 8 条线上产生有效的请求时，首先作一排队，找到最高优先级者响应。响应过程中，$EDACK_2$~$EDACK_0$ 三条线输出一个 3 位的二进制码表示目前正在服务的通道编号。

HOLD 82380 输出给 CPU 的总线请求信号。HLDA 是 CPU 回答的总线响应信号，82380 收到此信号后，即可接管系统总线。在整个 DMA 过程中，这一对信号始终保持有效。

EOP# DMA 过程结束时产生的一个双向信号。82380 首先输出一个 EOP# 信号通知正在 DMA 传送的通道，其 DMA 过程即将结束。对 82380 输入的 EOP# 信号则表示可以结束。

三、82380 的中断功能

82380 的中断控制器相当于包含三片 8259A 电路，分别称作为中断层(bank)A 中断层 B 和中断层 C。这三层串接起来产生一个总的中断请求信号 INT，接至 CPU 的 INTR 输入。它共有 5 个内部中断请求及 15 个外部中断请求输入端 IRQ#。每个外部中断请求端又可以扩展

接一片 8259A 作为从片。因此最多可以管理 15×8=120 个外部中断请求信号。

同一片 82380 的每个中断请求都可独立设置中断矢量，而不是像 8259A 那样各个中断矢量自动连续，另一方面，当外接 8259A 作为从片时，在中断响应周期里，从片的编码不是由 CAS_2～CAS_0 引线送入，而是通过数据线 D_7～D_0 传输。除此两点以外，每一层的中断控制器和单片的 8259A 无大差别。

15 个外部中断请求输入端的编号是 IRQ3#、IRQ9#、IRQ11#～IRQ23#。其中 IRQ3# 和 IRQ9# 有双重功能。

IRQ3# 和定时器 2 的输出 TOUT2# 接到一起引出片外。所以这条引线或者仅作为 IRQ3# 输入，或者仅作为 TOUT2# 输出，或者是用 TOUT2# 来产生 IRQ3# 请求。

IRQ9# 也可用作 DMA 控制的输入请求 DREQ4。

5 个内部中断请求的情况如下：

IRQ8# 接计数器 0，IRQ0# 接计数器 3。由这两个计数器的输出端 TOUT 上升沿触发中断请求信号。

IRQ1# 和 IRQ4# 用于内部的 DMA 控制。当 DMA 中基地址寄存器没有赋值时则产生 DMA 连锁中断请求 IRQ1#。而当软件 DMA 请求被清除后，则产生 DMA 终止计数中断请求 IRQ4#。

IRQ1.5# 是 82380 内中断层 A 比其它中断层添加的一个中断请求输入。由于它的中断优先级低于 IRQ1# 而高于 IRQ2#，故称为 IRQ1.5#. 当对中断控制器 A、B、C 三层中任何一层写入一个初始化命令字 ICW2 时，即产生 IRQ1.5# 中断，故称之为 ICW_2 写入中断请求。

还值得一提的是不算外部中断请求，又不算内部中断请求的 IRQ7#，它实际上是一种容错处理。若因为某种错误触发了中断，CPU 进入中断响应周期后，却又无法找到任何一个有效的中断请求，中断控制电路就自动产生 IRQ7# 中断矢量。

四、82380 的定时器

82380 内部有 4 个 16 位可编程间隔定时器，编号 0～3。每个定时器的功能与单片 8253 类似。四个定时器共用时钟信号 CLKIN。各定时器作用如下：

定时器 0 输出信号的上升沿产生中断请求 IRQ8#。在系统中，定时器 0 常用作 24 小时的时钟。

定时器 1 用作 DRAM 刷新控制。其输出端 TOUT1# 的上升沿送到刷新控制电路产生刷新请求信号。

定时器 2 的输出端 TOUT2# 和中断请求线 IRQ3# 相联引出。当它作为定时器输出时可驱动扬声器发声。

定时器 3 一方面在 82380 内部产生 IRQ0 中断请求，另一方面通过 TOUT3# 引出作为一个通用的外部信号。

第五节 数/模(D/A)和模/数(A/D)转换技术及其接口

A/D(模/数)及 D/A(数/模)转换技术广泛应用于计算机控制系统及数字测量仪表中。把模拟量信号转换成数字量的器件称为模/数转换器(简称 A/D 转换器)，而把数字量信号转换成模拟量信号的器件称为数/模转换器(简称 D/A 转换器)。

一、D/A 转换器

(一) D/A 转换器的工作原理

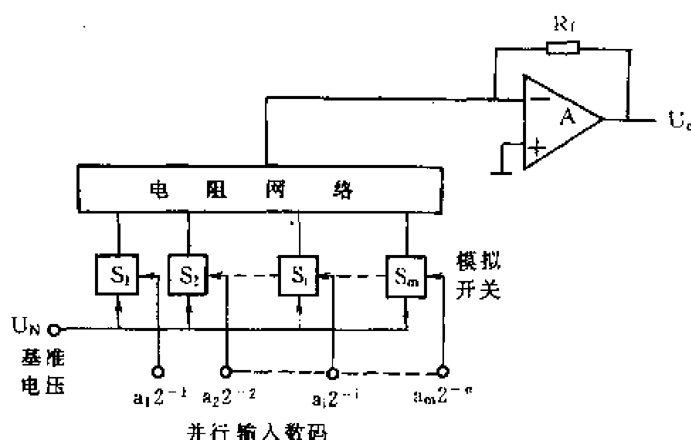


图 9-28 并行 D/A 转换器原理

D/A 转换器从工作原理上可分为并行 D/A 转换器及串行 D/A 转换器两种。并行 D/A 转换器的转换速度快，但电路复杂。随着微电子技术的发展，并行 D/A 转换器集成电路目前已大量生产，广为采用。

并行 D/A 转换器的位数与输入数码的位数相同，对应输入数码的每一位都设有信号输入端，用以控制相应的模拟切换开关，把基准电压 Un 接到电阻网络上。并行 D/A 转换器的原理见图 9-28。

电阻网络将基准电压转变为相应的电流或电压，在运算放大器的输入端进行相加。放大器的输出则反映了输入数码的大小。

如输入数码 $X_p=a_1 2^{-1}+a_2 2^{-2}+...+a_i 2^{-i}+...+a_n 2^{-n}$，则：

$$U_O=U_N X_p= U_N(a_1 2^{-1}+a_2 2^{-2}+...+a_i 2^{-i}+...+a_n 2^{-n})$$

$$=U_N\sum_{i=1}^{n}a_i 2^{-i}$$

其中，a_i 是 1 还是 0，取决于输入数码第 i 位是逻辑 1 还是逻辑 0. 如果 $a_i=1$，基准电压 UN 通过模拟切换开关加到电阻网络上如果 $a_i=0$ 模拟切换开关断开，基准电压 U_N 不能加到电阻网络上.

并行 D/A 转换器的转换速度很快，只要输入端加入数码信号，输出端立即有相应的模拟电压输出.

在并行 D/A 转换器中，常用的电阻网络是“T”形网络。12 位 T 形网络 D/A 转换器原理见图 9-29。它由 12 个串联分路开关，27 个精密电阻和一个运算放大器组成。电阻网络只用 R 或 2R 两种规格的电阻。电阻网络的输出接至运算放大器，若反馈电阻 R_f 的值为 3R，则总的输出电压 Uo 为：

$$U_O=-U_O\frac{R_f}{R_i}=-\frac{2}{3}U_N X_p\times\frac{3R}{2R}=-U_N X_p$$

式中 R_i——运算放大器的输入运算电阻，$R_i=2R$。

因此，当输入二进制码 X_p 为全 1，运算放大器输出为-$(1-1/2^{12})U_N$；当输入二进制码 x_p 为全 0，则运算放大器输出为 0。所以，D/A 转换的输出在 0～-$(1-1/2^{12})U_N$ 之间变动。

(二) D/A 转换器的主要技术指标

1. 分辨率　这是 D/A 转换器对微小输入量变化的敏感程度的描述，通常用数字量的位数来表示，如 8 位、10 位等。对一个分辨率为 n 位的转换器，能够分辨满刻度的 2_n 输入信号。

2. 稳定时间　指 D/A 转换器加上满刻度的变化(如全“0”变为全“1”)时，其输出达到稳

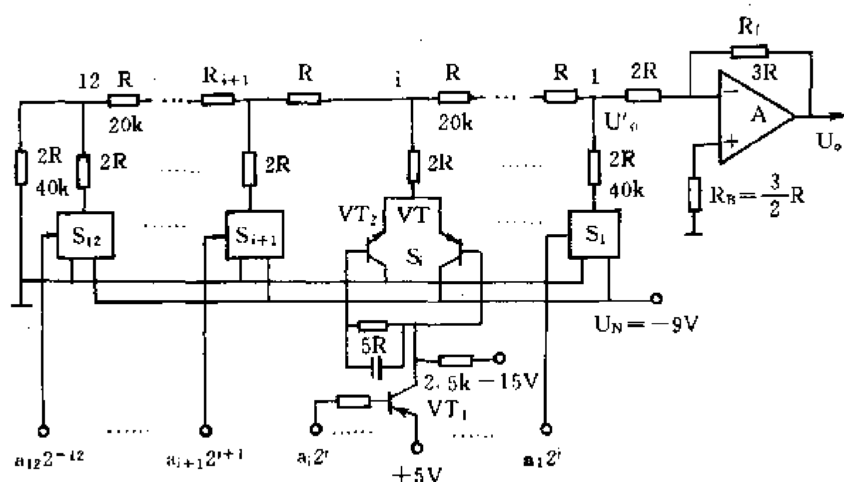


图 9-29 12 位 T 形网络 D/A 转换器原理图

定(一般稳定到与+1/2 LSB(最低有效位)值相当的模拟量范围内)所需的时间。一般为几十毫秒到几微秒。

3. 输出电平 不同型号的 D/A 转换器的输出电平相差较大。一般电压型的 D/A 转换器输出为 0~5 伏或 0~10 伏。电流型的 D/A 转换器,输出电流为几毫安至几安。

4. 绝对精度 对应于给定的满刻度数字量,D/A 实际输出与理论值之间的误差。该误差是由于 D/A 的增益变化、零点飘移和噪声等引起的。一般应低于 $2^{-(n+1)}$或 1/2 LSB。

5. 相对精度 在满刻度已校准的情况下，在整个刻度范围内对应于任一数码的模拟量输出与理论值之差。对于线性的 D/A 转换器,相对精度就是非线性度。有两种方法表示相对精度,一种是将偏差用数字量的最低位的位数 LSB 表示；一种是用该偏差相对满刻度的百分比表示。

6. 线性误差 相邻两个数字输入量之间的差应是 1LSB,即理想的转换特性应是线性的。在满刻度范围内,偏离理想的转换特性的最大值称线性误差。

7. 温度系数 在规定的范围内,相应于每变化 1℃,增益、线性度、零点及偏移(对双极性 D/A)等参数的变化量。温度系数直接影响转换精度。

(三) 典型 D/A 转换器芯片

1. D/A 转换器芯片 DAC 0832

(1) DAC 0832 芯片的内部结构

DAC 0832 是美国国家半导体公司生产的八位集成电路芯片,其逻辑结构框图见图 9-30。由图可见,该芯片内部有两个数据缓冲寄存器,八位输入寄存器和八位 DAC 寄存器。其转换结果以一组差动电流 I_{out1}和 I_{out2}输出。DAC 0832 的八位输入寄存器的输入端 DI_7~DI_0 可直接与 CPU 的数据线相连接。两个数据缓冲寄存器的工作状态分别受$\overline{LE_1}$和$\overline{LE_2}$控制。当$\overline{LE_1}=0$(低电平)时,八位输入数据寄存器的输出跟随输入而变化。当$\overline{LE_1}$由低电平变为高电平,即$\overline{LE_1}=1$ 时,输入数据立即被锁存。同理,八位 DAC 寄存器的工作状态受$\overline{LE_2}$的控制。

(2)DAC 0832 芯片的引脚及其功能

DAC 0832 共有 20 条引脚,见图 9-31,各引脚定义如下：

DI_7~DI_0 D/A 转换器的数字量输入信号。其中,DI_0 为最低位,DI_7 为最高位。

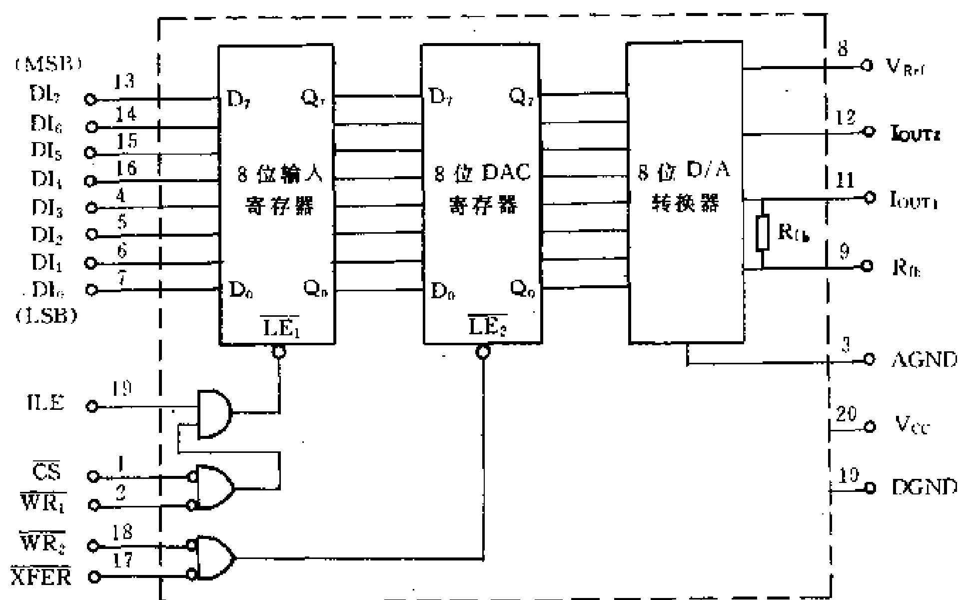


图 9-30 DAC 0832 芯片内部逻辑结构

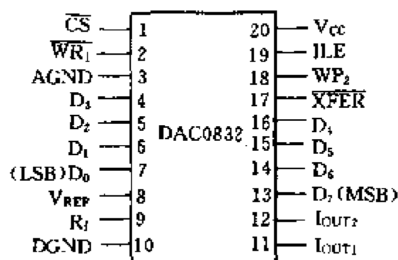


图 9-31 DAC 0832 引脚图

$\overline{CS}$ 片选输入信号，低电平有效。

$\overline{WR_1}$ D/A 转换器的数据写入信号 1，低电平有效。

ILE 输入寄存器的允许信号，高电平有效。ILE 信号和$\overline{CS}$、$\overline{WR1}$共同控制选通输入寄存器。当$\overline{CS}$、$\overline{WR_1}$均为低电平，而 ILE 为高电平时，$\overline{LE_1}=0$，输入数据立即被送至八位输入寄存器的输出端；当上述三个控制信号任一个无效时，$\overline{LE_1}$变高，输入寄存器将数据锁存，输出端呈保持状态。ILE＝0 时，$\overline{LE_1}$也无效，输出端便不随 D 端而变化。

$\overline{XFER}$ 从输入寄存器向 DAC 寄存器传送 D/A 转换数据的控制信号，低电平有效。

$\overline{WR_2}$ DAC 寄存器的选通信号，低电平有效。当$\overline{XFER}$和$\overline{WR_2}$同时有效时，输入寄存器的数据被装入 DAC 寄存器，并同时启动一次 D/A 转换。

Vcc 芯片电源，其值可在＋5～＋15V 之间，典型值为＋15V。

AGND 模拟信号地。

DGND 数字信号地。

R_{fb} 内部反馈电阻引脚，用来外接 D/A 转换器输出增益调整电位器。

V_{Ref} D/A 转换器的基准电压，其范围可在-10～＋10V 之间选定。

I_{out1} D/A 转换器输出电流 1，当输入数字为全"1"时，其值最大；全"0"时，其值最小。

I_{out2} D/A 转换器输出电流 2，它与 I_{out1}的关系如下：

$$I_{out1}+I_{out2}=V_{Ref}/R\cdot(1-\frac{1}{2^8})$$

(3). DAC 0832 的工作方式

DAC 0832 有三种工作方式：

a. 双缓冲工作方式

DAC 0832 芯片内有两个数据寄存器，在双缓冲工作方式下，CPU 要对 DAC 芯片进行两步写操作：① 将数据写入输入寄存器。② 将输入寄存器的内容写入 DAC 寄存器。其连接方式是：把 ILE 固定为高电平，$\overline{WR_1}$、$\overline{WR_2}$均接到 CPU 的$\overline{IOW}$，而$\overline{CS}$和$\overline{XFER}$分别接到两个端口的地址译码信号。

双缓冲工作方式的优点是 DAC 0832 的数据接收和启动转换可异步进行。可以在 D/A 转换的同时，进行下一数据的接收，以提高模出通道的转换速率，可实现多个模出通道同时进行 D/A 转换。

b. 单缓冲工作方式

此方式是使两个寄存器中任一个处于直通状态，另一个工作于受控锁存器状态。一般是使 DAC 寄存器处于直通状态，即把$\overline{WR_2}$和$\overline{XFER}$端都接数字地。此时，数据只要一写入 DAC 芯片，就立刻进行数模转换。此种工作方式可减少一条输出指令，在不要求多个模出通道同时刷新模拟输出时，可采用此种方式。

c. 直通工作方式

将$\overline{CS}$、$\overline{WR_1}$、$\overline{WR_2}$和$\overline{XFER}$引脚都直接接数字地，ILE 引脚为高电平时，芯片即处于直通状态。此时，八位数字量一旦到达 DI_7～DI_0 输入端，就立即进行 D/A 转换而输出。但在此种方式下，DAC 0832 不能直接和 CPU 的数据总线相连接，故很少采用。

(4). DAC 0832 的主要技术指标

电流建立时间	1μs
单电源	+5～+15V
V_{Ref}输入端电压	±25V
分辨率	8 位
功率耗能	200mW
最大电源电压 V_{DDK}	17V

2. D/A 转换器芯片 DAC 1210

(1)DAC 1210 芯片内部逻辑结构

DAC 1210 是美国国家半导体公司生产的 12 位 D/A 转换器芯片，是智能化仪表中常用的一种高性能的 D/A 转换器。

由图 9-32 可见，其逻辑结构与 DAC 0832 类似，所不同的是 DAC 1210 具有 12 位的数据输入端，且其 12 位数据输入寄存器由一个 8 位的输入寄存器和一个 4 位的输入寄存器组成。两个输入寄存器的输入允许控制都要求$\overline{CS}$和$\overline{WR_1}$为低电平，但 8 位输入寄存器的数据输入还要求 $B_1/\overline{B_2}$端为高电平。

(2) DAC 1210 芯片的引脚及其功能

DAC1210 的引脚见图 9-33 所示。

$\overline{CS}$　片选信号，低电平有效。

$\overline{WR1}$　写控制信号 1，低电平有效。此信号为高电平时，两个输入寄存器都不接收新数据。当此信号有效时，与 $B_1/\overline{B_2}$ 配合起控制作用。

AGND　模拟地。

DI_{11}～DI_0　12 位数字量输入，DI_0 为最低位。

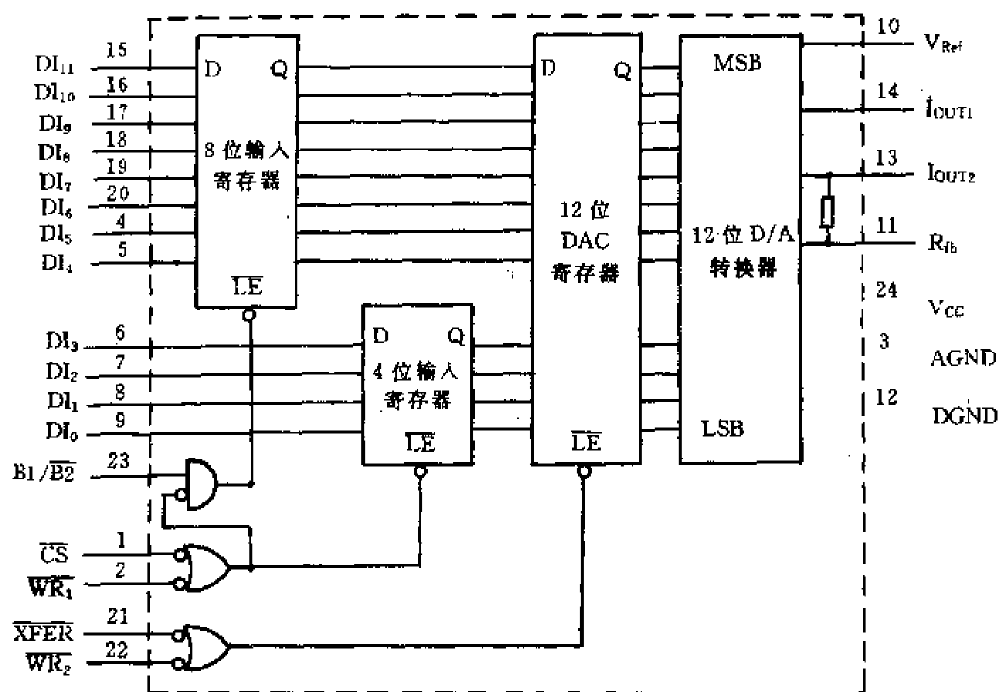


图 9-32　DAC 1210 内部逻辑结构图

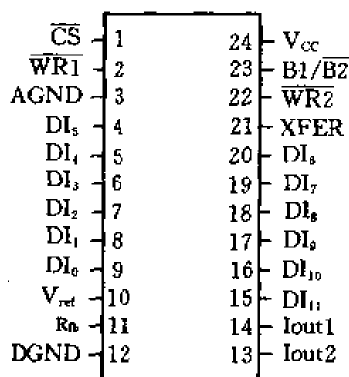


图 9-33　DAC 1210 引脚图

V_{Ref}　参考电压。

V_{fb}　外部放大器的反馈电阻接线端。

DGND　数字地。

I_{out1}　D/A 电流输出端 1。

I_{out2}　D/A 电流输出端 2。

$\overline{XFER}$　数据转换控制信号，低电平有效，与$\overline{WR_2}$配合使用。

$\overline{WR_2}$　写控制信号 2，低平有效。此信号有效时，$\overline{XFER}$信号才起作用。

$B_1/\overline{B_2}$字节控制。此端为高电平时，12 位数字同时送入输入锁存器。此端为低电平时，只将 12 位数字量的低 4 位送到 4 位输入寄存器中。

(3) DAC 1210 主要技术指标

输入　　12 位数字量

输出　　模拟量电流 I_{out1}和 I_{out2}

电流稳定时间　1μs

功耗　　20mW

工作电压　+5～+15V 电源

参考电压　可工作在+10～-10V 范围内

输入逻辑电平　与 TTL 兼容

芯片内有锁存器，可直接连到 CPU 的数据总线上。

工作环境温度范围　-40℃～+85℃

三种输入方式：双缓冲、单缓冲和直接输入三种方式

（四）D/A 转换器与 CPU 的接口应用举例

D/A 转换器与微处理器间的信号连接包括三部分，即数据线、控制线和地址线。

微处理器的输出数据要传送给 D/A 转换器，首先要把数据总线的输出信号连接到数/模转换器的数据输入端。微处理器因要进行各种信息的处理，其数据总线上的数据总是不断变化，输出给 D/A 转换器的数据，只是在执行输出指令的几微秒中出现在数据总线上。而 D/A 转换器要求数字量并行输入，且其输入数据要在一定时间内保持稳定，以满足精度要求。因此微处理器数据总线上输出的数据必须用一个锁存装置锁存起来，这个锁存装置就是 D/A 转换器与 CPU 的数据接口。

图 9-34 给出了 DAC 1210 与 IBM PC 标准总线的连接图。

DAC 1210 的 12 位数据线与 8 位数据总线相连接时，可将 DAC 1210 输入线的高 8 位 DI_{11}～DI_4 与 IBM PC 的数据总线 DB_7～DB_0 相连，而其低 4 位 DI_3～DI_0 也接至 IBM PC 数据总线的 DB_7～DB_4 上，12 位的数据应由两次写入操作完成。图 9.34 中，设 DAC 1210 占用了 0450H～0452H 三个端口地址，为使两次数据输入口地址是先偶（0450H）后奇（0451H），与编程习惯一致，将 AB0 地址线经反相驱动器接至 $B_1/\overline{B}_2$ 端。

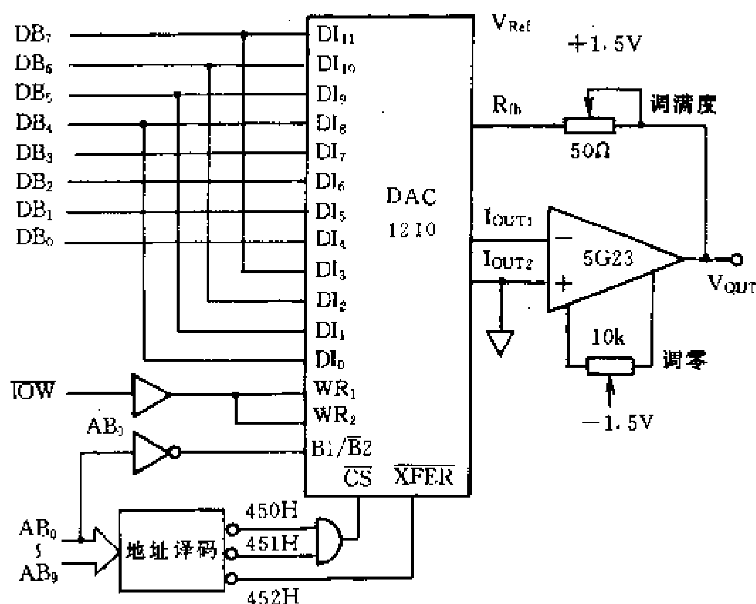


图 9-34　DAC1210 与 8 位微处理器的连接

由于 DAC 1210 中的 4 位寄存器的 LE 端只受 $\overline{CS}$ 和 $\overline{WR_1}$ 控制，而其 8 位输入寄存器也受 $\overline{CS}$ 和 $\overline{WR_1}$ 控制，故两次写入操作均使 4 位寄存器的内容更新。因此正确的操作步骤是：先使 $B_1/\overline{B}_2$ 端为高电平，先写入高 8 位寄存器；再使 $B_1/\overline{B}_2$ 端为低电平，以保护 8 位寄存器已写入的内容，同时进行第二次写入操作。虽然第一次写入操作时，4 位寄存器中也写入某个值，但第二次写入操作后，此值便被更改为所需值。

下面的程序段为图 9-34 中完成一次转换输出的程序。

```
                          ;设 BX 寄存器中低 12 位为待转换的数字量
START:  MOV   DX, 0450H   ;DAC 1210 的基地址
```

```
MOV   CL, 04
SHL   BX, CL       ;BX 中的 12 位数左移 4 位
MOV   AL, BH       ;高 8 位数→AL
OUT   DX, AL       ;写入高 8 位
INC   DX           ;修改 DAC 1210 端口地址
MOV   AL, BL       ;低 4 位数→AL
OUT   DX, AL       ;写入低 4 位
INC   DX           ;修改 DAC 1210 端口地址
OUT   DX, AL       ;启动 D/A 转换
HLT
```

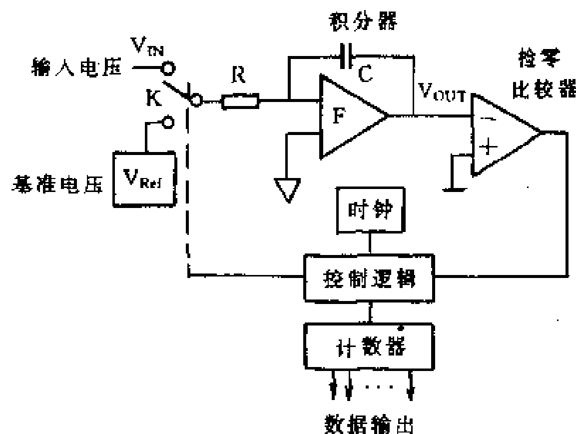
二、A/D 转换器

(一) A/D 转换器的工作原理

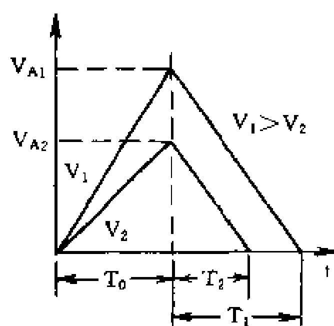
1. 双积分型 A/D 转换器

双积分式也称二重积分式，其原理框图如图 9-35(a)，具体工作过程如下：

转换开始后，首先使积分电容完全放电，并将计数器清零。然后使开关 K 先接通输入电压 V_i，积分器对 V_i 定时积分，当定时 T_0 到时，控制逻辑使 K 合向基准电压 V_{Ref}端，并让计数器开始计数，此时积分电容开始反向积分(放电)，至输出电压为 0 时，比较器翻转，控制计数器停止计数。图 9-35(b)为两次积分的波形图。可以看出，在正向积分时间 T_0 固定的情况下，反向积分时间 T_i(图(b)中的 T_1 或 T_2)正比于输入电压 V_i，T_i 的数值可由计数器得到。



(a)原理框图



(b)积分器波形器

图 9-35 双积分型 A/D 的工作原理

第一阶段，对模拟输入电压 V_i 积分时，输出 V_A 为：

$$V_A=\frac{1}{RC}\int_0^{T_0} V_i dt=\frac{1}{RC}T_0V_i$$

即积分器的输出与模拟输入电压 V_i 的平均值成正比。

第二阶段，积分器对 V_{Ref}进行反向积分，直至积分器输出为 0，即

$$V_{out}=V_A-\frac{1}{RC}\int_0^{T_0}V_{Ref}dt=0$$

即

$$V_A-\frac{1}{RC}T_1V_{Ref}=0$$

$$V_A=\frac{1}{RC}T_1V_{Ref}$$

由上可知：

$$\frac{1}{RC}T_0V_i=\frac{1}{RC}T_1V_{Ref}$$

$$T_1=\frac{V_i}{V_{Ref}}T_0$$

由于 T_0、V_{Ref}为已知的固定常数，因此反向积分时间 T_1 与输入模拟电压 V_i 在 T_0 时间内的平均值成正比。输入电压 V_i 愈高，V_A愈大，T_1 就愈长。在 T_1 开始时刻，控制逻辑同时打开计数器的控制门开始计数，直到积分器恢复到零电平时，计数停止。则计数器所计的数字即正比于输入电压 V_i 在 T_0 时间内的平均值，于是完成了一次 A/D 转换

由于双积分型 A/D 转换是测量输入电压 V_i 在 T_0 时间内的平均值，所以对常态干扰(串模干扰)有很强的抑制作用，尤其对正负波形对称的干扰信号，抑制效果更好。

双积分型的 A/D 转换器电路简单，抗干扰能力强，精度高 ，这是突出特点。但转换速度慢，常用的 A/D 转换芯片的转换时间为毫秒级。因此适用于模拟信号变化缓慢，采样速率要求较低，而对精度要求较高，或现场干扰较严重的场合。例如在数字电压表中常被采用。

2. 逐次逼近型 A/D 转换器

逐次逼近型(也称逐位比较式)的 A/D 转换器，应用比积分型更为广泛 ，其原理框图如图 9-35 所示，主要由逐次逼近寄存器 SAR，D/A 转换器，比较器以及时序和控制逻辑等部分组成。它的实质是逐次把设定的 SAR 寄存器中的数字量经 D/A 转换后得到电压 Vc，与待转换模拟电压 V_x 进行比较。比较时，先从 SAR 的最高位开始，逐次确定各位的数码应是“1”还是“0”，其工作过程如下：

转换前，先将 SAR 寄存器清零。转换开始时，控制逻辑电路先设定 SAR 寄存器的最高位为“1”，其余位为“0”，此试探值经 D/A 转换成电压 V_c，然后将 V_c 与模拟输入电压 V_x 比较。如果 $V_x \geq V_c$，说明 SAR 最高位的“1”应予保留；如果 $V_x < V_c$，说明 SAR 该位应予清零。然后再对 SAR 寄存器的次高位置“1”，依上述方法进行 D/A 转换和比较。如此重复上述过程，直至确定 SAR 寄存器的最低位为止。过程结束后，状态线改变状态，表明已完成一次转换。最后，逐次逼近寄存器的内容就是与输入模拟量 V_X 相对应的二进制数字量。A/D 转换器的位数 N 决定于 SAR 的位数和 D/A 的位数。图 9-36(b)表示四位 A/D 转换器的逐次逼近过程。转换结果能否准确逼近模拟信号，主要取决于 SAR 和 D/A 的位数，位数越多，越能准确逼近模拟量，但转换所需的时间也越长。

(二) A/D 转换器的主要技术指标

1. 分辨率

分辨率表示转换器对微小输入量变化的敏感程度，通常用转换器输出数字量的位数来表示。例如，对 8 位 A/D 转换器，其数字输出量的变化范围为 0～255，当输入电压满刻度为 5V 时，转换电路对输入模拟电压的分辨能力为 5V/255＝19.6 mV。目前常用的 A/D 转换集成芯

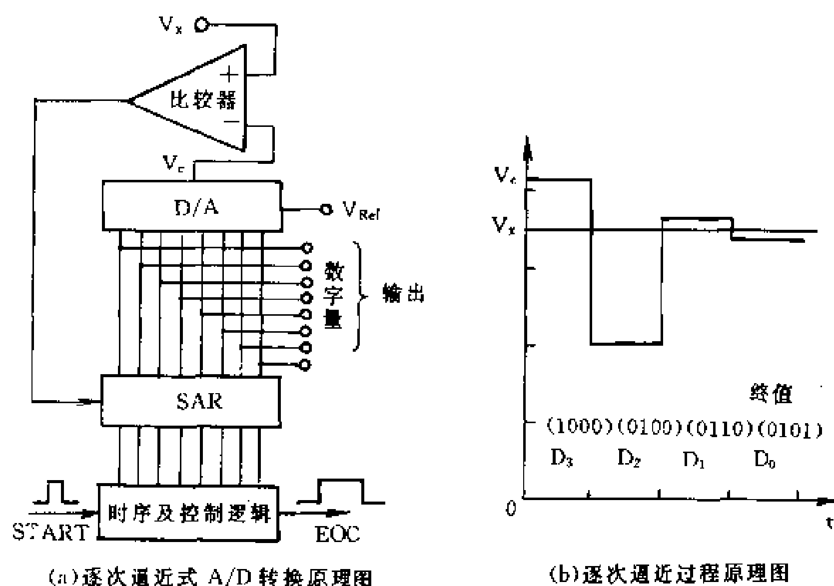


图 9-36 逐次逼近型 A/D 转换器原理

片的转换位数有 8 位、10 位、12 位和 14 位等。

2. 精度

A/D 转换器的精度是指与数字输出量所对应的模拟输入量的实际值与理论值之间的差值。A/D 转换电路中与每个数字量对应的模拟输入量并非是单一的数值，而是一个范围△，△的大小，在理论上取决于电路的分辨率，例如，对满刻度输入电压为 5V 的 12 位 A/D 转换器，△为 1.22mV。定义△为数字量的最小有效位 LSB。但在外界环境的影响下与每一数字输出量对应的输入量实际范围往往偏离理论值△。

精度通常用最小有效位 LSB 的数值来表示。目前常用的 A/D 转换集成芯片的精度为 1/4～2LSB。

3. 转换时间

完成一次 A/D 换所需要的时间，称为 A/D 转换电路的转换时间。目前，常用的 A/D 转换集成芯片的转换时间约为几个 μs～200μs。在选用 A/D 转换集成芯片时，应综合考虑分辨率、精度、转换时间，使用环境温度以及经济性等诸因素。12 位 A/D 转换器适用于高分辨率系统；陶瓷封装 A/D 转换芯片适用于-25～+85℃ 或-55～125℃，塑料封装芯片适且于 0～70℃。

4. 温度系数和增益系数

这两项指标都是表示 A/D 转换器受环境温度影响的程度。一般用每摄氏温度变化所产生的相对误差作为指标，以 ppm/℃为单位表示。

5. 对电源电压变化的抑制比

A/D 转换器对电源电压变化的抑制比(PSRR)用改变电源电压使数据发生±1LSB 变化时所对应的电源电压变化范围来表示。

(三) 典型 A/D 转换器芯片

1. A/D 转换器芯片 ADC 0809

(1) ADC 0809 芯片的内部结构

ADC 0809 的内部逻辑结构如图 9-37 所示，其结构分为四个部分。

(1) 模拟输入部分

有 8 路单端输入的多路开关和地址锁存与译码逻辑，可由三位地址输入 ADDA、ADDB、ADDC 编码选择 8 路中的一路输入(这三个地址输入信号可锁存)。

(2) 变换器部分

主要由四部分组成：

a. 控制逻辑　提供转换器的时钟 CLK 和起动信号 START。转换完成时，发出转换结束信号 EOC(End Of Convert)，高电平有效。

b. 逐位逼近寄存器 SAR(8 位)

c. 比较器

d. 电阻网络

(3) 三态输出缓冲器(8 位)

(4) 基准电压输入端 REF(+)和 REF(-)

它们决定了输入模拟电压的最大值和最小值. 通常把 REF(+)接到 V_{CC}(+5 伏)电源上，REF(-)接到地端 GND。当然，REF(+)和 REF(-)也可不接到 V_{CC}和 GND 上，但加在此两个输入端的电压 $V_{Ref}(+)$和 $V_{Ref}(-)$必须满足以下条件：

$$0 \leqslant V_{Ref}(-) < V_{Ref}(+) \leqslant V_{CC}$$

且

$$\frac{V_{Ref}(+)+V_{Ref}(-)}{2} = \frac{1}{2}V_{CC}$$

(2) ADC 0809 芯片的引脚及其功能

ADC 0809 芯片的外部引脚如图 9-38 所示，各引脚功能如表 9-4 所示。

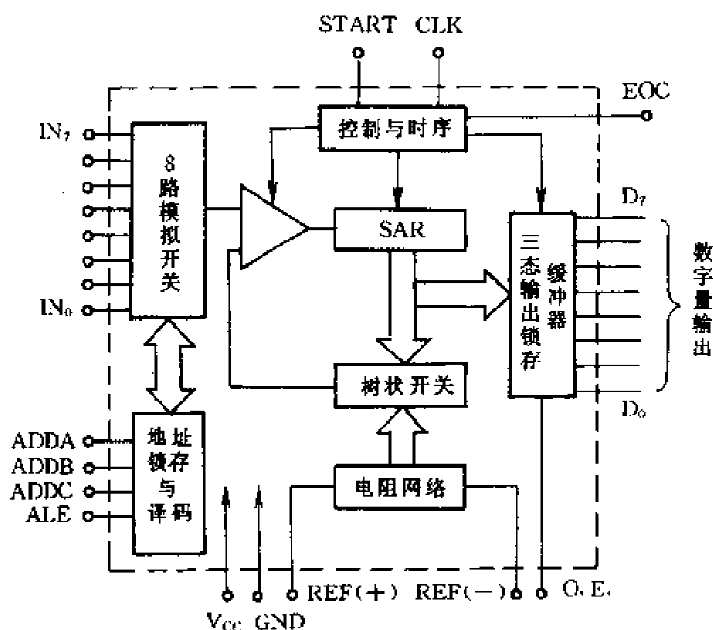


图 9-37　ADC 0809 内部逻辑结构图

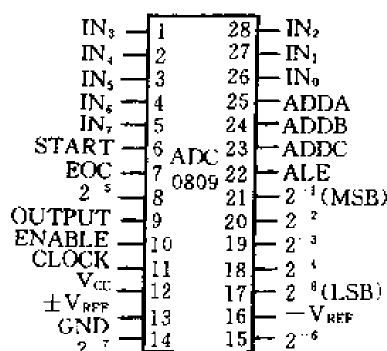


图 9-38　ADC 0809 引脚图

(3)ADC 0809 芯片的主要技术指标

电源电压	6.5V
分辨率	8 位
时钟频率	640kHz
转换时间	100μs

未经调整误差　　1/2LSB 和 1LSB
模拟量输入电压范围　　0～5V
功耗　　15mW

表 9-4　ADC 0809 引脚功能

符　号	引脚号	功　能
IN_0～IN_7	26～28,1～5	为 8 个通道模拟量输入线
ADD A		多路开关地址选择线。A 为最低位,C 为最高位,通常分别接在地址
ADD B	25～23	
2^{-8}～2^{-1}	17,14,15,8, 18～21	8 位数字量输出结果
ALE	22	地址锁存有效输入线。该信号上升沿把 ADDA、ADDB、ADDC 三选择线的状态锁存入多路开关地址寄存器中
START	6	启动转换输入线。该信号上升沿清除 ADC 的内部寄存器而在下降沿启动内部控制逻辑,开始 A/D 转换工作
EOC	7	转换完成输出线,当 EOC 为 1 时表示转换已完成
CLOCK	10	转换完成时时钟输入线。其频率不能高于 640kHz,当频率为 640kHz 时,转换速度约 100μs
OUTPUT ENABLE	9	允许输入线。在 OE 为"1"时,三态输出锁存器脱离三态,把数据送往总线
$V_{Ref}(+)$、$V_{Ref}(-)$	12,16	参考电压输入线
V_{CC}	11	接+5V
GND	13	接地

2. A/D 转换芯片 AD574A

(1)AD574A 的内部结构

AD574A 是 12 位逐次逼近式的 ADC。转换时间为 25～35μs。片内有数据输出寄存器,并有三态输出的控制逻辑。其运行方式灵活,可进行 12 位转换,也可作 8 位转换;转换结果可直接 12 位输出,也可先输出高 8 位,后输出低 4 位。可直接与 8 位,12 位和 16 位的 CPU 接口。输入可设置成单极性,也可设成双极性。片内有时钟电路,无需加外部时钟

AD574A 适用于对精度和速度要求较高的数据采集系统和实时控制系统。

AD574A 的内部逻辑结构框图如图 9-39 所示。

(2) AD574A 芯片的引脚及其功能

如图 9-40 所示为 AD574 芯片的引脚图,各主要引脚的含义如下:

DB_{11}～DB_0　输出数据线。DB_{11}为 MSB 位(最大有效位),DB_0 为 LSB 位(最小有效位)。转换后的数字量经片内输出三态缓冲器后,由这些数据线输出。

$\overline{CS}$　片选信号,低电平有效。

CE　片允许信号,高电平有效。

$R/\overline{C}$　读/启动转换信号。该引脚为低电平时,表示启动转换;高电平时,表示可输出数据。

A_0(4 脚)和 12/8(2 脚)　配合用于控制转换数据长度是 12 位或 8 位,以及数据输出的格式(是 12 位一次输出还是先输出高 8 位,后输出低 4 位)。A_0=0,表示启动一次 12 位转换;

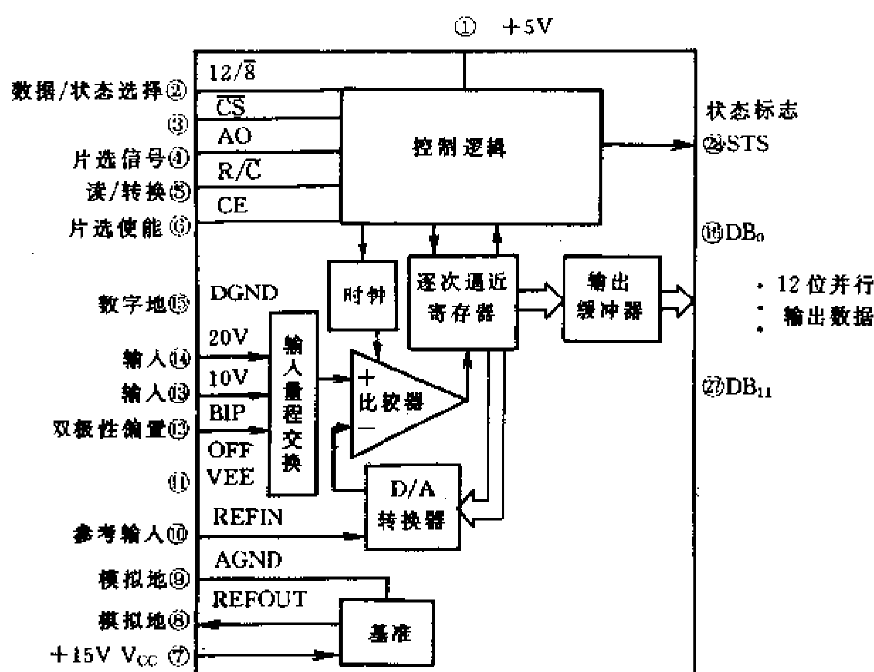


图 9-39 AD574A 内部逻辑结构图

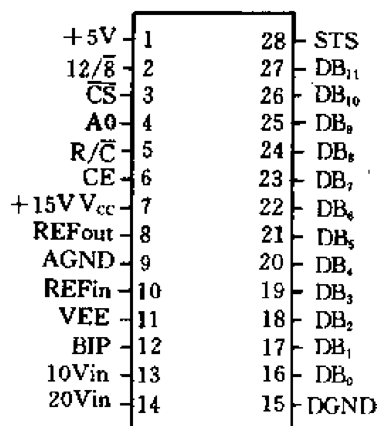


图 9-40 AD574 引脚图

$A_0=1$，表示启动一次 8 位转换。

STS(28 脚) 转换状态输出端，转换过程中呈现高电平，转换一结束立即返回到低电平。

$10V_{in}$(13 脚) 此引脚的模拟量输入范围是 0～+10V；如果接成双极性工作方式，可以是-5V～+5V。

20V(14 脚) 此引脚的模拟量输入范围是 0～+20V；若为双极性，输入范围可为-10V～+10V。

BIP OFFSET(12 脚) 此引脚的连接方式，与模拟量是单极性还是双极性有关。

REF_{in}(10 脚) 参考电压输入端。

REF_{out}(8 脚) 参考电压输出端。

(3) AD574 的主要技术指标

AD574A 有 6 个等级。AD574J、K、L 专用于在 0～+70℃ 温度范围内。AD574S、T、U 专用于-55～+125℃范围内。

其中，AD574ATD 性能指标如下：

(1)分辨率	12 位	
(2)非线性误差	±1LSB	
(3)模拟输入：	双极性	±5V，±10V
	单极性	0～+10V，0～+20V
(4)供电电源	V_{LOGIC}	+4.5～+5.5V
	V_{CC}	+13.5～+16.5V
	V_{DD}	+13.5～+16.5V

(5)内部参考电平　　10.00± 0.1(max)V

(6)转换时间　　15～35 μs

(7)存放温度　　-65～150℃

(四) A/D 转换器与 CPU 的接口应用实例

图 9-41 为 AD574A 与 8088CPU 的接口框图,具体讨论如下:

1. 模拟量输入部分

如图 9-41 所示为双极性接线方式,模拟量输入范围可达 10V,由采样保持器输入。

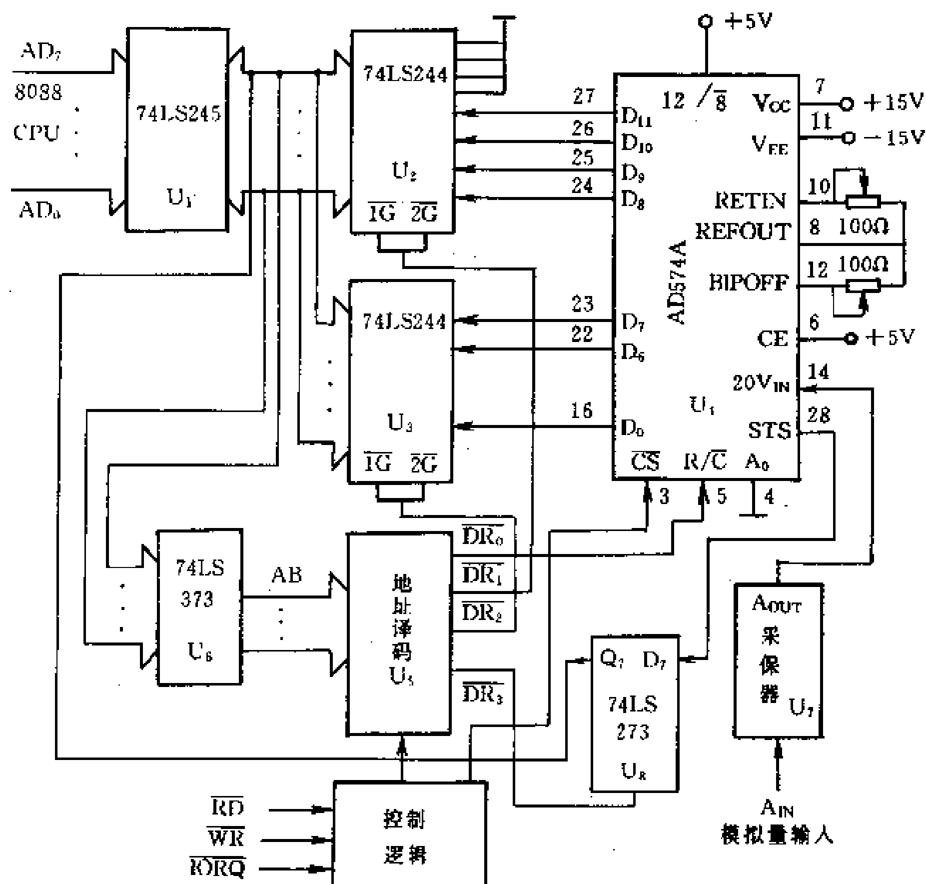


图 9-41 AD574A 与 8088CPU 接口框图

2. 数据总线接口部分

8088 CPU 通过 U_1、U_2 和 U_3 与 AD574A 接口。U_1 采用 74LS245 芯片,是 8 位同相三态收发器,由于 8088 的 AD_7～AD_0 是数据线与低 8 位地址线分时复用的,U1 既作为 AD574A 转换后数字量的输入缓冲器,也作为地址和控制命令输出的驱动器。A/D 转换后的 12 位数据,并行输出,高 4 位送至 U_2,低 8 位送至 U_3 缓存,然后分两次通过 U_1 读入。因此 U_2、U_3 可选用单方向的八位同相三态缓冲器 74LS244,且将 U_2 输入端的高 4 位接地,使读入 CPU 的 A/D 转换后的数字量范围为 0000H～07FFH。同时为保证 CPU 能分两次正确读入 AD574A 转换后的结果,U_2、U_3 必须用不同的端口地址去选通。

3. AD574A 控制逻辑选择

(1)$\overline{CS}$和 CE 信号

根据 AD574A 的时序要求,启动 A/D 转换和读转换结果都必须在使能信号 CE=1 和$\overline{CS}$

=0 的情况下进行。因此可将 CE 端接+5V，而$\overline{CS}$可由$\overline{RD}$、$\overline{WR}$信号组合控制，即只要对 AD574A 进行$\overline{RD}$或$\overline{WR}$操作，$\overline{CS}$都有效。

(2) A_0 和 2/8 选择

将 A_0 接地，而 $12/\overline{8}$接+5V，表示此芯片用于 12 位的转换，同时读出时是 12 位数字量并行输出。

(3) 启动转换和读数据控制

$R/\overline{C}$信号可由地址译码器的输出 DR_0 控制，只要对 DR_0 地址端口进行一次写操作，便可使 $R/\overline{C}$由高变低，有一个大于 200ns 的下降沿，便可启动一次 A/D 转换。

AD574A 的工作状态由 STS 信号输出，可将 STS 信号经 U_8 和 U_1 读至 AD_7，以查询 A/D 是否转换完毕，可否读入数据。也可利用 STS 信号去控制采样保持器的工作模式，以简化接线。

读入转换结果时，只要分别对 DR_1 和 DR_2 地址端口进行读操作，此时 $R/\overline{C}=1$，即 AD574A 处于读状态，便可将转换后的 12 位数据分别按高位字节和低位字节，读至 CPU 中。

综上所述，对图 9-41 的接口电路，可写出在查询方式下 AD574A 的转换程序如下：

```
START:  MOV   DX, DR0
        OUT   DX, AL          ;使 R/C=0,启动 A/D 转换
        MOV   DX, DR3
TEST:   IN    AL, DX          ;读 STS 状态
        AND   AL, 80H
        JNZ   TEST            ;未转换完,再测试
        MOV   DX, DR1
        IN    AL, DX          ;转换完,读入高 4 位
        MOV   BL, AL          ;BH←高 4 位
        MOV   DX, DR2
        IN    AL, DX          ;读入低 8 位
        MOV   BL, AL          ;BL←低 8 位
        HLT                   ;暂停
```

思考题与习题

1. 8253/8254 芯片的计数与定时功能有哪些区别？
2. 8253/8254 芯片的 6 种工作方式各自有哪些特点？
3. 定时器各通道的 GATE 信号有哪些作用？在不同工作方式下其功能相同吗？
4. 选择题：

(1) 8253/8254 计数器工作期间，CPU 重新对定时器编程是(　)。

A. 任何情况下禁止的

B. 任何情况下允许的，且影响当前计数

C. 任何情况下允许的，且不影响当前计数

D. 任何情况下允许的，且影响程序随方式而变

(2) 计数器的门控信号 GATE 的作用是(　)。

A. 由低变高使计数器开始计数

B. 由高变低使计数器停止计数

C. 处于常低禁止计数器工作

D. 处于常高允许计数器工作

(3) 对定时器 3 个通道的编程顺序是( )。

A. 完全随机的,但必须设置好一个通道后再设置另一个通道

B. 完全固定的,从通道 0 开始到通道 2

C. 完全随机的,但必须先初始化方式字

D. 完全随机的,但必须先预置计数初值

(4) 对定时器发声编程与对 8255 端口 B 发声编程比较,其优点主要表现在( )。

A. 音调频率范围大　　　B. 编程方便

C. 程序通用性好　　　D. 声音持续时间可变

(5) 定时器通道 0 用于日时钟中断,通道 2 用于扬声器发声音调。有时,发声编程要屏蔽中断,其原因是( )。

A. 通道 0 工作要影响通道 2 输出波形

B. CPU 执行日时钟中断使音调持续时间拉长

C. CPU 执行日时钟中断要改变音调频率

D. 日时钟频繁中断使发声程序无法执行

5. 异步通信有哪些特点? 同步通信有哪些特点?

6. RS-232C 标准是为何种设备之间通信制定的? 它主要包含哪些特性?

7. 查询 I/O 方式下异步通信编程一般步骤是怎样的?

8. 中断 I/O 方式下异步通信编程一般步骤是怎样的?

9. 选择题:

(1) 异步通信传输信息时,其特点是( )。

A. 通信双方不必同步

B. 每个字符的发送是独立的

C. 字符之间的传输时间长度应相同

D. 字符发送速率由波特率确定

(2) 同步通信传输信息时,其特点是( )。

A. 通信双方必须同步

B. 每个字符的发送不是独立的

C. 字符之间的传输时间长度可不同

D. 字符发送速率由数据传输率确定

(3) 异步通信接收方采用 16 倍频的发生器时钟作为接收信号时钟,其目的是( )。

A. 取样精度高　　　B. 接收速度可以加快

C. 取样对准峰值信号　　　D. 避免起始位由噪声产生

(4) 在编制异步传输查询接收程序时,最容易发生的传输错误是( )。

A. 接收到无效字符　　　B. 超越错

C. 奇偶校验错　　　D. 帧格式错

(5) 既然在数据传输率相同的情况下,那么,又说同步字符传输速度要高于异步字符

传输其原因是（ ）。

A. 发生错误的概率少　　B. 附加位信息总量少

C. 双方通信同步　　D. 字符之间无间隔

(6) RS-232C 定义的 EIA 标准规定连接器的物理结构是（ ）。

A. DB-25 型　　B. DB-15 型

C. DB-9 型　　D. 未作定义

(7)RS-232C 定义的 EIA 电平范围对输出信号和输入信号之所以允许有 2V 的压差，其目的是（ ）。

A. 增加传输距离　　B. 减小波形失真

C. 克服线路损耗　　D. 提高抗干扰能力

10. 填空题：

(1) USART 是英文（ ）的缩写。

(2) 8251A 在异步通信时最大波特率可达到（ ），此时 CLK 的频率至少为（ ）Hz，该器件在同步通信时，最大波特率可达到（ ）。

(3) 只有在（ ）信号到来之后，或者最先写入（ ）后，才能将方式控制字写入 8251A。

11. 填空题：

(1) PC 系列机的一个并行接口具有（ ）个缓冲输出端。CPU 可用 IN 或 OUT 指令对其读写. 另外它还有（ ）个稳态输入端，CPU 用 IN 指令可对其读出。

(2) 并行接口逻辑内部具有三个设备端口：（ ），CPU 对它们的访问有五种操作：（ ）。

12. 选择题：

(1) 通常，PC 系列机配置的针式打印机是（ ）。

A. 串行口串行打印机　　B. 串行口并行打印机

C. 并行口串行打印机　　D. 并行口并行打印机

(2) 对 8255A 芯片，若端口 A 为方式 1 输出，端口 B 为方式 1 输入，PC4 和 PC5 为输入，则控制字代码应是（ ）。

A. 0AEH　　B. 0BEH

C. 0A1H　　D. 0B1H

(3) 8255A 芯片选方式 0 时输入输出的工作特点是：只（ ）信号有效，就有数据传送. 此外，端口工作方式控制字的最高位 D_7 必须是（ ），这是此控制字的特征标志位。

A. $\overline{WR}$, 1　　B. $\overline{RD}$, 0

C. $\overline{WR}$, 0　　D. $\overline{RD}$, 1

E. $\overline{WR}$或$\overline{RD}$, 1　　F. $\overline{WR}$或$\overline{RD}$, 0

13. 用 8255A 的 A 口选方式 1 作输入口，而 B 口选方式 1 作输出口，假设控制字寄存器口地址为 0FBH，写出相应的初始化程序段。

14. 什么叫 D/A 的分辨率？什么叫相对转换精度？

15. 用带两级数据缓冲器的 D/A 转换器时，为什么有时要用三条输出指令才完成 16 或

12 位数据转换？

16. 使用 DAC0832 进行数/模转换时，有哪两种方法可对数据进行锁存？

17. 在数字量和模拟量并存的系统中，地线连接时要注意什么问题？

18. 什么叫模/数转换精度？什么叫转换率？什么叫分辨率？

19. 双积分式 A/D 转换的原理是什么？

20. 比较计数式、双积、和逐次逼近式 A/D 转换的优缺点。

21. 设计一个电路并画出软件流程以实现 A/D 转换，软件流程中要体现逐次逼近法。

第十章　高性能微处理器

第一节　80286 微处理器

一、概述

1982 年 1 月 Intel 公司推出的 80286 是比 8086/8088 更先进的 16 位微处理器芯片，其内部操作和寄存器都是 16 位。该芯片集成了 13.5 万个晶体管，以 68 引线四列双插式封装，不再使用分时复用线，具有独立的数据线 16 条，地址线 24 条，时钟频率为 8～10MHz。

80286 片内具有存储管理和保护机构。80286 对存储器采用分段的管理办法，每段最大为 64KB，并支持虚拟存储器。这样的存储器管理方法使 80286 能可靠地支持多用户系统。

80286 有两种工作方式，即实地址方式和虚地址保护方式。在实地址方式下就是一个快速的 8086；在虚地址保护方式下，80286 可寻址 16MB(2^{24})物理地址，并能提供 1000MB(2^{30})的虚拟地址空间。80286 可以配接浮点处理器 80287。

80286 具有 8086/8088 的全部功能。8086/8088 的汇编语言程序不做任何修改即可在 80286 上运行。

二、80286 的结构

1. 功能结构

80286 具有对多用户和多任务系统的处理能力，其 CPU 的内部方框图如图 10-1 所示。

由图 10-1 可看到，80286 比 8086 增加了指令部件 IU(Instruction Unit)，把 8086 的总线接口单元 BIU 分成了地址部件 AU(Address Unit)、指令部件 IU 和总线部件 BU(Bus Unit)。这样，就增加了这些部件的并行操作，提高了吞吐率，加快了处理速度。80286 内部并行操作的情况如图 10-2 所示。

2. 80286 的编程结构

(1)寄存器组

80286 的寄存器组如图 10-3 所示。

其中，通用寄存器(包括数据寄存器和基址变址寄存器)、段寄存器与 8086 的完全一样。但是状态和控制寄存器有些变化，主要表现在新增加了三个标志位和新增加了机器状态字 MSW(Machine Status Word)寄存器。

(2)标志字寄存器

80286 的标志字寄存器如图 10-4 所示。

除了 8086 中的 9 个标志位以外还增加了三个标志位。

I/O 特权标志：IOPL(位 12、13)用以指定 I/O 操作处于 0～3 特权层中的哪一层。

嵌套标志：NT(位 14)若 NT＝1，表示当前执行的任务嵌套于另一任务中，执行完该任务后，要返回到原来的任务中去；否则 NT＝0。

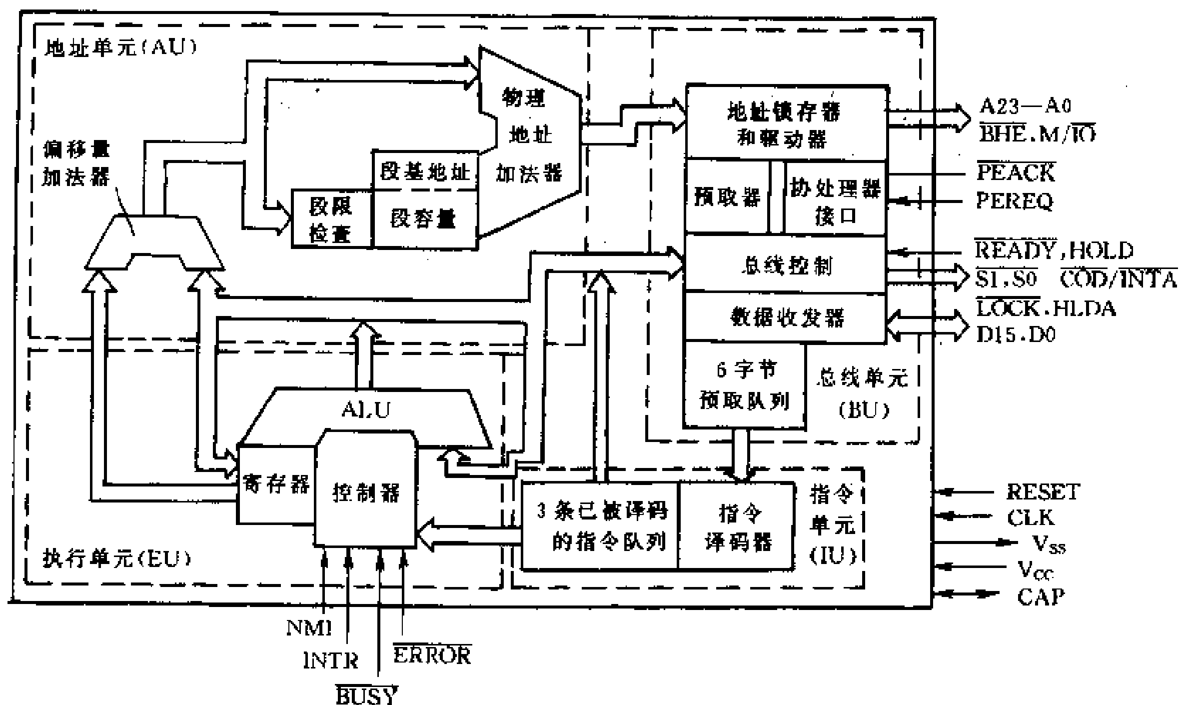


图 10-1 80286 的内部方框图

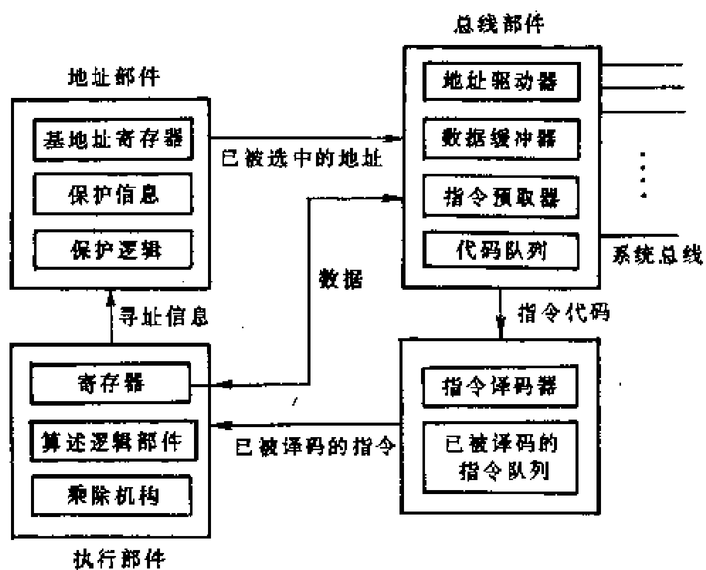


图 10-2 80286 内部的并行操作

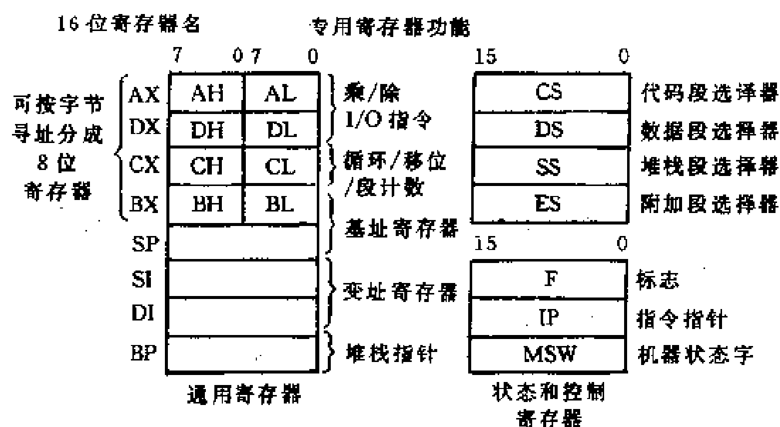


图 10-3 80286 的基本寄存器组

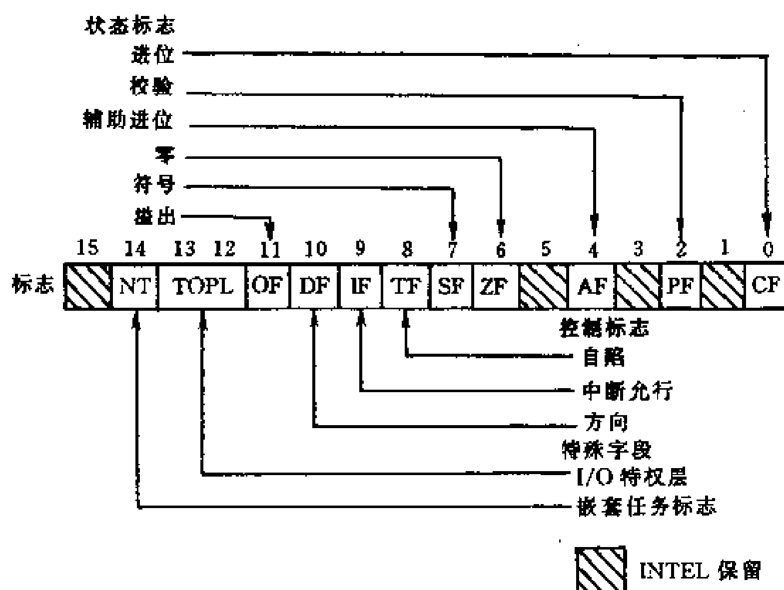


图 10-4 80286 的标志字寄存器

(3)机器状态字

80286 的机器状态字 MSW 如图 10-5 所示。

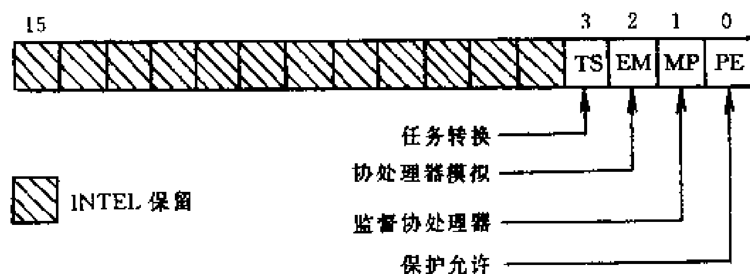


图 10-5 80286 的机器状态字

这是一个 16 位寄存器,目前只用它的低 4 位。其功能如表 10-1 所示。

表 10-1　机器状态字位功能表

位	名　称	功　　能
0	PE	保护方式允许：把 80286 置入保护方式，并且除 RESET 外不能被清除
1	MP	监督协处理器：允许 WAIT 指令引起协处理器不存在异常(7 号)
2	EM	模拟协处理器：当 ESC 指令允许模拟一个协处理器时，将引起协处理器不存在异常(7 号)
3	TS	任务转换：表示下一条若使用协处理器指令将会引起异常 7，允许用软件测试当前协处理器处理的上下文是否属于当前任务

其中，除了最低位是使 80286 进入保护方式外，其它三位起控制协处理器接口的作用。在系统复位后，MSW 置为 FFF0H，CPU 处于实地址方式。指令 LMSW(装载 MSW)和 SMSW (存 MSW) 可以在实地址方式下装入和存放 MSW。TS、MP 和 EM 的功能编码如表 10-2 所示。

表 10-2　MSW 中控制协处理器的位的编码

TS	MP	EM	意 义	造成异常的指令
0	0	0	仅是 80286 的实地址方式，为 RESET 后的最初编码，80286 的操作与 8086、8088 的相同	无
0	0	1	没有协处理器可供使用，软件将模拟它的功能	ESC
1	0	1	没有协处理器可供使用，软件将模拟它的功能。当前协处理器的上下文可以属于另一个任务。	ESC
0	1	0	协处理器存在	无
1	1	0	协处理器存在。当前协处理器的上下文可以属于另一个任务。关于 WAIT 的异常，允许软件检测来自前一个协处理器操作期间的错误	ESC 或 WAIT

三、80286 的引脚

80286 的芯片引出脚如图 10-6 所示。它突破了通常的 CPU 的 40 个引出脚，为 68 脚的双列直插式封装的引脚。由于引出脚数量的增多，80286 中已经不采用引出脚的分时复用，而是具有独立的 24 条地址线和 16 条数据线。

下面对 80286 引出脚的主要功能作些扼要的介绍。其中 I 表示输入，O 表示输出，I/O 表示既可用于输入也可用于输出。

(1) CLK　I 系统时钟

此时钟为 286 系统提供基本定时。通常是一个 16MHz 的信号，在 80286 的内部被除以 2 变为 8MHz 的处理器时钟。

(2) D_{15}～D_0　I/O 数据总线

CPU 用这些数据线，在存储器、I/O 和中断响应读周期输入数据；在存储器和 I/O 写周期输出数据。数据总线是高电平有效，在总线保持响应期间，浮空到第三态 OFF。

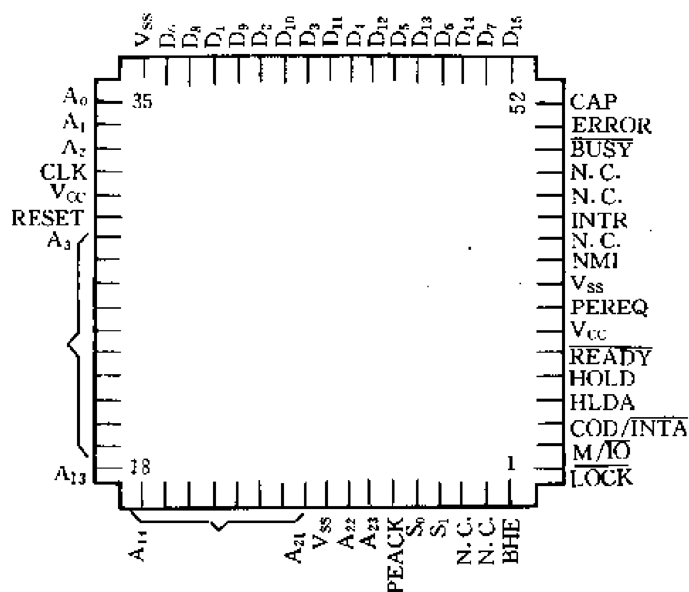


图 10-6　80286 引出脚图

(3) A_{23}～A_0　地址总线

CPU 利用这些地址线输出存储器地址和 I/O 端口的地址。当数据要在引出脚 D_7～D_0 上传送时，A_0 是低电平。在 I/O 传送时，只用 16 位地址，故 A_{23}～A_{16} 为低电平。地址总线是高电平有效，在总线保持响应期间，浮空到第三态 OFF。

(4) $\overline{BHEO}$　0 总线高位允许线

该引出脚的功能和 8086 的 $\overline{BHE}$ 引出脚功能基本相同，它表示数据在数据线的字节 D_{15}～D_8 上进行传送。指定为数据总线高 8 位的设备，常常使用 $\overline{BHE}$ 作为片选信号。$\overline{BHE}$ 和 A_0 的编码如表 10-3 所示。

表 10-3　$\overline{BHE}$ 和 A_0 的编码

$\overline{BHE}$	A_0	功　　能
0	0	字传送
0	1	在数据总线高 8 位进行字节传送(D_{15}～D_8)
1	0	在数据总线低 8 位进行字节传送(D_7～D_0)
1	1	保留

(5) $\overline{S}_0$、$\overline{S}_0$　0 总线周期状态

这两个信号的变化，标识着一个总线周期的开始，并且与 M/$\overline{IO}$ 和 CON/$\overline{INTA}$ 信号一起定义是什么样的总线周期。$\overline{S}_1$、$\overline{S}_0$ 是低电平有效，在总线保持响应期间浮空到第三态 OFF。$\overline{S}_1$、$\overline{S}_0$ 与 COD/$\overline{INTA}$，M/$\overline{IO}$ 对总线周期的定义如表 10-4 所示。

表 10-4　80286 总线周期状态定义

CON/INTA	M/$\overline{IO}$	$\overline{S}_1$	$\overline{S}_2$	启动的总线周期
0	0	0	0	中断响应
0	0	0	1	保留
0	0	1	0	保留
0	0	1	1	不是一个状态周期
0	1	0	0	若 A_1=1，则暂停；否则停机
0	1	0	1	读存储器数据
0	1	1	0	写存储器数据
1	1	1	1	不是一个状态周期
1	0	0	0	保留
1	0	0	1	读 I/O

（续）

CON/$\overline{INTA}$	M/$\overline{IO}$	$\overline{S_1}$	$\overline{S_2}$	启动的总线周期
1	0	1	0	写 I/O
1	0	1	1	不是一个状态周期
1	1	0	0	保留
1	1	0	1	读存储器指令
1	1	1	0	保留
1	1	1	1	不是一个状态周期

(6) M/$\overline{IO}$　0 存储器、IO 选择线

CPU 用此信号区分对存储器或 IO 设备的访问，若此信号为高电平，则正处于一个存储器周期或一个暂停/停机(halt/shutdown)周期；若为低电平，则正处于一个 I/O 周期或中断响应周期。M/IO 信号在总线保持响应期间，浮空到第三态 OFF。

(7) COD/$\overline{INTA}$　0 代码/中断响应

CPU 用此信号，在存储器操作时，区分是读取指令还是读取数据；在 I/O 操作时，区分是读 I/O 还是中断响应。在总线保持响应期间，浮空到第三态 OFF。

(8) $\overline{LOCK}$　0 总线锁定

此信号有效，表示在当前的总线周期后，系统中其他的总线主设备不能取得系统总线的控制权。$\overline{LOCK}$信号可以直接由"LOCK"这个指令前缀使其有效，或由 80286 的硬件在执行存储器 XCHG 指令，中断响应或描述符表访问期间自动产生。$\overline{LOCK}$信号是低电平有效，并且在总线保持响应期间，浮空到第三态 OFF。 2

(9) $\overline{READY}$　0 总线准备就绪

这是 CPU 的一个输入信号，是由外部(CPU 的外部)输给 CPU 的结束某个总线周期操作的信号。CPU 的总线周期，只有在接收到有效的 READY 信号后才结束，否则则总线周期将无限地扩展。READY 是一个低电平有效的信号，它的建立和保持时间必须满足系统规定的要求。在总线保持响应期间，READY 信号被忽略。

(10) HOLD　I、HLDA　0 总线保持请求和保持响应

当系统中的其他总线主设备请求总线控制权时，向 CPU 发出有效的 HOLD 作号。当允许总线转让时，在 CPU 接收到有效的 HOLD 信号后，就把它的总线驱动器浮空到第三态，并输出有效的总线保持响应信号 HLDA，这样，就进入了总线保持响应状态。只要 HOLD 信号有效，请求总线的主设备一直可以保持对总线的控制权。当 HOLD 变为无效时，80286 将撤消 HLDA，并重新取得总线的控制权，这样，就终止了总线保持响应状态。这些信号高电平有效。

(11) INTR　I 中断请求

这是由外部设备向 CPU 发出的，请求 80286 挂起当前正在执行的程序，为一个外部的请求服务。CPU 是否响应此请求，受 80286 标志字中的中断允许位的控制，若此控制位被清除，则中断请求就被屏蔽。当 80286 响应一个中断请求时，它执行两个中断响应周期，在第二个中断响应总线周期，80286 从数据总线的低 8 位上读 取一个识别中断源的中断问题。为了确保程序中断，INTR 必须保持有效，直到第一个中断响应周期完成。CPU 是在每个处理器周期的开头采样 INTR，而在当前指令结束时才能响应，故 INTR 的有效高电平至少要持续两个处理

器周期，以便能在下一条指令之前实现中断。INTR 信号是电平触发，高电平有效。

(12) NMI　I 非屏蔽中断请求

这是外部的一种紧急的中断请求，它不受 80286 标志字中的中断允许位的控制。当它有效时，CPU 必须在当前指令执行完以后响应。INTEL 规定此类中断请求的中断向量为 2。在响应这类中断请求时，CPU 执行中断响应周期，从中断向量 2 直接取出中断服务程序的入口，转入中断服务。NMI 输入是高电平有效，可以与系统时钟不同步，并且是在内部同步后由边沿触发的。为了正确地识别，NMI 信号在有效前必须至少有 4 个系统时钟周期是低电平。并在有效后，至少保持 4 个系统时钟周期的高电平。

(13) PEREQ　I、$\overline{\text{PEACK}}$　O 协处理器操作数请求和响应

80286 能把存储器管理和保护能力扩展到协处理器。PEREQ 输入请求 80286 为一个协处理器执行一个数据操作数传送。当被请求的操作数正在传送时，80286 用有效的 PEACK 信号通知协处理器。PEREQ 信号高电平有效，$\overline{\text{PEACK}}$信号低电平有效。

(14) $\overline{\text{BUSY}}$　I、$\overline{\text{ERROR}}$　I　协处理器忙或出错

$\overline{\text{BUSY}}$信号向 80286 指示协处理器的操作情况。有效的$\overline{\text{BUSY}}$输入，将停止 80286 程序在 WAIT 和一些 ESC 指令上的执行，直到$\overline{\text{BUSY}}$变成无效（高电平）。在等待$\overline{\text{BUSY}}$变成无效期间，80286 可以被中断。有效的 ERROR 输入，将使 80286 在执行 WAIT 或一些 ESC 指令时，实现协处理器的中断。这些输入是低电平有效。

(15) RESET　I　　系统复位

系统复位信号，高电平有效。不论在什么时间、什么情况下，80286 都可用有效的 RESET 信号来重新初始化。RESET 信号的高电平，至少应保持 16 个系统时钟周期，在 RESET 有效期间，停止 80286 内部的任何操作，并使它的引出脚处于表 10-5 所示的状态。

表 10-5　RESET 有效期间输出引出脚的状态

引出脚的状态	引　出　脚　名
1（高电平）	$\overline{S_0}$、$\overline{S_1}$、$\overline{\text{PEACK}}$、$A_{23}\sim A_0$、$\overline{\text{BHE}}$、$\overline{\text{LOCK}}$
0（低电平）	M/$\overline{\text{IO}}$、COD/$\overline{\text{INTA}}$、HLDA
三态(OFF)	$D_{15}\sim D_0$

80286 的操作是在 RESET 引出脚上的信号由高电平转换为低电平后开始的。RESET 的高电平到低电平的转换，必须同步于系统时钟。从电源接通到第一个取代码的总线周期之间，为了 80286 内部的初始化，大约需要 50 个时钟周期。

与系统时钟同步的 RESET 从高电平到低电平转换后，将在系统时钟的下一个高电平到低电平的转换时，开始一个新的处理器周期。

(16) V_{SS}　I 系统地。

(17) V_{CC}　I 系统电源，+5V 电源。

(18) CAP　I 衬底滤波电容器。

在此引出脚和地之间，必须接一个 0.047μF±20%的电容器。该电容器用来滤内部衬底偏压发生器的输出，该电容器允许有 1μA 的漏电流。

为了使 80286 能正确操作，衬底偏压发生器必须把这个电容充电到它的工作电压。在 Vcc 和 CLK 到达它们额定的 AC 和 DC 参数以后，这个电容的充电时间为 5ms（最大）。在这

期间，RESET 可以用来防止 CPU 产生假动作。在这以后，通过产生同步于系统时钟的 RESET LOW 脉冲，80286 处理时钟可以与另一个时钟节拍同步。

四、80286 的实地址方式

80286 的实地址方式与 8086 的工作方式几乎是完全相同的。下面将其特点扼要地介绍一下：

1. 存储器容量

80286 有 24 条地址引线 $A_0 \sim A_{23}$。但在实地址方式下，只有 $A_0 \sim A_{19}$是有用的，可寻址 1M 字节的连续空间，高地址线 $A_{20} \sim A_{23}$是不用的，目的是与 8086 兼容 。

2. 存储器寻址

80286 在实地址方式的寻址与 8086 中是一样的，任何一个实际物理单元的地址由段地址和段内偏移量两部分组成。

段地址通常是由某个段寄存器的 16 位数左移 4 位，低 4 位总是 0 而形成的 20 位段基地址。因此，段地址总是从 16 字节的倍数开始，全部段的上限是 64K 字节，这些段都可以被读、写或执行。

段内偏移量总是 16 位，可由各种寻址方式来形成。

20 位段基地址与 16 位段内偏移量相加(进位被忽略)形成了某一物理单元的实际地址，如图 10-7 所示。

3. 保留的存储单元

在实地址方式下，80286 保留的两个存储区域如图 10-8 所示。

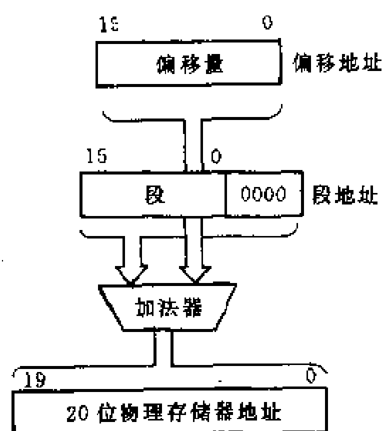


图 10-7　80286 实地址方式的地址计算

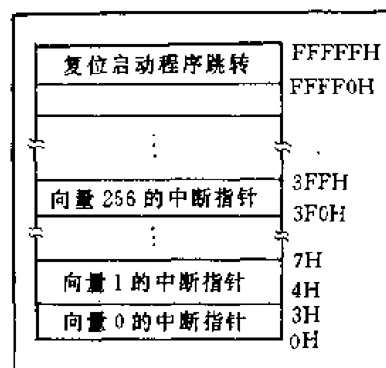


图 10-8　80286 在实地址方式下保留的存储单元

在存储器的高端 0FFFF0H～0FFFFFH 保留的是系统的初始化区。与 8086 中一样，80286 在复位以后，CS＝FFFF，而 IP＝0000，故初始化从地址 0FFFF0H 开始执行。通常，在初始化区中存放一条段间无条件转移指令，转移到系统的初始化程序的入口处。

在存储器的低端 00000H～003FFH，安放的是中断向量表，为 256 个中断(或异常) 的服务程序提供入口。在实地址方式下，中断向量表中的 4 个字节与 8086 中的一样，是中断服务程序入口的段基地址(在地址计算时自动左移 4 位)和段内偏移量。但在虚地址保护方式下，由于存储空间更大了，就更为复杂一些。

4. 中断

80286 在实地址方式下的中断指令和外部中断与 8086 是一样的。由于寻址错误所引起的

异常和中断如表 10-6 所示

表 10-6 实地址方式寻址中断

功　　能	中断号	相　关　的　指　令	返回指令前的地址吗?
中断表限太小异常	8	INT 向量不在表限内	返回
协处理器段越出中断	9	存储器操作数要超出偏移量 0FFFFH 的 ESC	不返回
段越出异常	13	用偏移量 0FFFFH 进行存储器引用,或试图越过段尾执行	返 回

异常将使 CPU 停留在执行引起异常的指令之前的状态上(PUSH 、POP、PUSHA 或 POPA 除外)。

五、80286 的虚地址保护方式

在实地址方式下操作的 80286 只相当于一个快速的 8086,并没有真正发挥 80286 的作用。只有虚地址保护方式才是 80286 的真正特色,才能充分发挥 80286 的作用。

为了满足多用户多任务系统的要求,80286 的主要特点是:存储管理和特权与保护。

1. 存储器寻址

在虚地址保护方式下,80286 的 24 位地址全能发挥作用,故其直接寻址为 $2^{24}=16M$ 字节。通过集成在片内的保护机构,能对每个任务提供最大可达 1000 兆字节的虚拟存储器空间。虚拟地址空间,比物理地址空间大,这是由 80286 的保护机构作为支柱的。每当一条指令引用任何不能映射到物理存储器的地址时,处理器就会引起一个可重新启动的指令异常,在进行必要的调进和调出以后,指令就能正确执行。

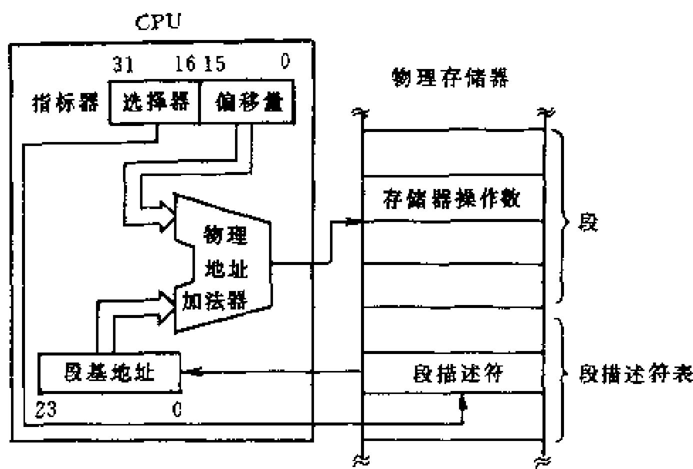


图 10-9 虚地址保护方式的存储器寻址

和实地址方式一样,物理存储器的地址也是由两部分组成的:段基地址和段内偏移量。但在虚地址保护方式下的段基地址应该是 24 位而不是实地址方式下的 16 位。而段内偏移量与实地址方式时相同,是由各种寻址方式所决定的 16 位值。80286 中的段 寄存器为了与 8086 兼容是 16 位的,且 80286 中没有 24 位的寄存器,如何存放 24 位的段基地址呢?

把程序中可能用到的各种段(如代码段、数据段、堆栈段、附加段)的段基地址和相应的特性集合在一起形成一张表,称为描述符表,存放在存储器的某一区域。于是在虚地址保护方式下的各个段寄存器中的内容,不再是段基地址,而是一个参数,用这个参数从描述符表中取出相应的描述符(包括此段的 24 位段基地址及相应的特性)。故在虚地址保护方式下的存储器寻址如图 10-9 所示。

由段寄存器中的参数,从描述符表中取出相应的描述符,就找到了此段基地址,与 16 位偏

移量相加形成要寻址单元的物理地址。

80286 采用这种称为描述符的数据结构，在存储器寻址的过程中提供了一个间接层。有了这个间接层，80286 的存储器管理机构和保护机构就有了活动的舞台。例如操作系统利用这种间接层，就能够把某一个段放在实存中的任意位置，而不需要调整程序的地址常数，因此，用这种方法，既保持了实地址方式中的基本寻址过程，又使得 80286 省去了为使用虚存而重新编写应用程序的需要。

2. 描述符

为了虚地址保护方式下的寻址，使用了好几种不同的描述符。

(1) 选择器

在虚地址保护方式下的段寄存器是一个选择器，它由 3 个字段组成，如图 10-10 所示。

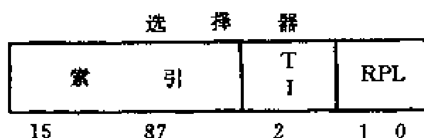


图 10-10 选择器格式

a、位 0、1 构成了选择器特权(数 0-3)，表示所请求的特权层 RPL(Requested Privilege Level)。

b、位 2 是一个表指示器 TI(Table Indicator)，表示此选择器选择哪一个描述符表。TI=0，选择全局描述符表(GDT)；TI=1，选择局部描述符表。

c、位 3～15 形成描述符表的入口索引，可以寻址 8K 个描述符。

(2) 代码段和数据段描述符

如前所述，80286 中用描述符这样的数据结构来实现寻址。不同的段就有不同的描述符，此外，80286 的控制转移和任务转换也涉及描述符。故 80286 中有多种不同的描述符。80286 有关于代码、堆栈和数据段的段描述符，也有关于特殊系统数据段和控制转移操作的系统控制描述符。

每个描述符的长度为 8 个字节。在代码段和数据段描述符中，除了该段的基地址之外，还包含了其他的段属性，主要有：

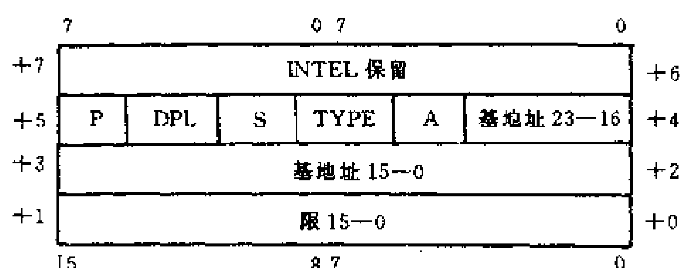
1. 段长度的界限(可为 1 到 64K 字节)；
2. 该段的访问权(只读、读/写、只执行和执行/读)；
3. 此段是否在物理存储器中(对虚拟存储器系统而言)；
4. 描述符特权层。

若对该段的访问违反了由段描述符所规定的属性，将会引起一个异常或中断。

代码和数据段描述符的格式如图 10-11 所示。一个描述符共 8 个字节，存储的最低两个字节规定了该段的最大长度。接着的 3 个字节(24 位)规定了该段的基地址。然后是一个称为访问权的字节，它规定了此段的特权级和属性。最后两个字节 80286 中不用，保留为 80386 及以后设计的芯片用。

访问权字节中的第 4 位 S 用以区分段的性质，S=1 则此段为代码段或数据段，S=0 则为系统控制描述符。第 7 位 P 用以确定此段是否存在于物理存储器中。第 5、6 两位为 DPL 位，规定了此段的特权等级，其值为 0～3。第 0 位 A 确定此段是否被访问过。余下的第 3、2、1 位规定了此段的类型，第 3 位 E 区分是代码段还是数据段，E=0 为数据段，E=1 为代码段。在这两种不同段中第 2、1 位的意义有些不同，已在图 10-11 中说明。

图 10-12 是一个数据段描述符的例子。



位	名　称	功　能	
7	Present (P)	P=1 该段在物理存储器中 P=0 该段不在物理存储器中,基地址和限没用	
6.5	Descriptor Privilege Level (DPL)	该段的特权属性。在特权测试中使用	
4	Segment Descriptor (S)	S=1 为代码段或数据段描述符 S=0 非段描述符	
3	Executable (E)	E=0 为数据段描述符	数据段
2	Expansion Direction (ED)	ED=0 向上生长段,偏移量必须≤限 ED=1 向下生长段,偏移量必须>限	数据段
1	Writeable (W)	W=0 数据段不能写入 W=1 数据段可以写入	数据段
3	Executable (E)	E=1 为代码段描述符	代码段
2	Conforming (C)	C=0 当 CPL≥DPL 时,代码段只能被执行	代码段
1	Readable (R)	R=0 代码段不能读 R=1 代码段可以读	代码段
0	Accessed (A)	A=0 该段尚未被访问过 A=1 段选择器已被装入段寄存器或已被测试指令使用过	

图 10-11　代码段和数据段描述符

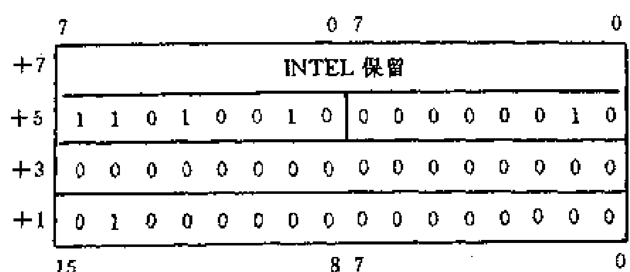


图 10-12　一个数据段描述符的例子

先看访问权字节的第 S 和 E 位,S=1,E=0,则为一个数据段描述符。P=1,此段在物理存储器中。DPL=10,即此段的特权级为 2。由段的类型场可知,此段是向上生长的可以写入的段,此段尚未被访问过(A=0)。段的基地址为 020000H,段的界限为 16K。

一个代码段描述符的例子如图 10-13 所示。

先分析访问权字节的 S 和 E 位,S=1 和 E=1 则为代码段。P=1,此段在物理存储器。DPL=01,则此段的特权级为 1。由于第 2 位 C=0,则当 CPL≥DPL,即当 CPL ≥ 1 时,该代码段只能执行(CPL 的意义在后面介绍)。第 1 位 R=0,此段不能读。最低位 A=1,此段已被访问过。该段的基地址为 820000H,段的界限为 8K。

(3) 系统控制描述符

除了代码段和数据段描述符外，在保护方式下的 80286 还定义了系统控制描述符。这些系统控制描述符定义了在保护环境下的特殊的系统数据段和转移机构。所谓特殊的系统数据段是指含有描述符表的段（局部描述符表描述符）和含有任务的执行状态的段（任务状态段描述符）。保护环境下 80286 的控制转移是通过一些门来实现的。

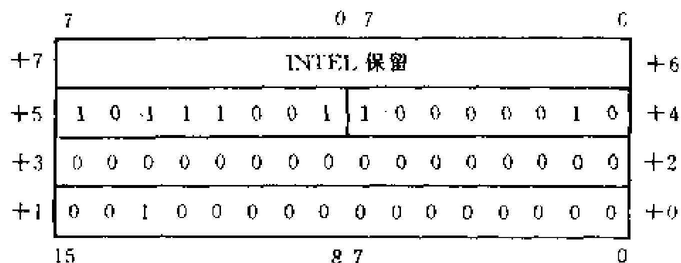
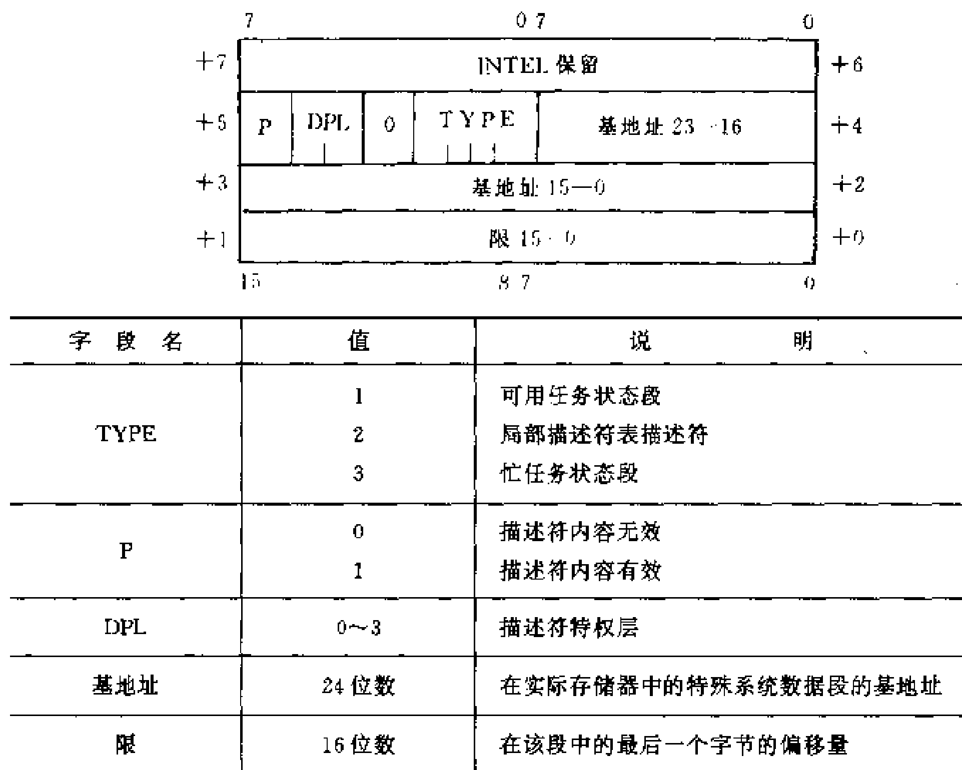


图 10-13　一个代码段描述符的例子

特殊系统数据段描述符的格式和字段定义如图 10-14 所示。



字段名	值	说明
TYPE	1 2 3	可用任务状态段 局部描述符表描述符 忙任务状态段
P	0 1	描述符内容无效 描述符内容有效
DPL	0～3	描述符特权层
基地址	24 位数	在实际存储器中的特殊系统数据段的基地址
限	16 位数	在该段中的最后一个字节的偏移量

图 10-14　特殊系统数据段描述符

此类段描述符与代码数据段描述符的最大区别在于访问权字节。其第 4 位即 S 位＝0，说明为非代码数据段描述符。其最高位 P，第 5、6 位的 DPL 的意义与代码数据段的相同。区别在于类型场，其具体意义已在图 10-14 中说明。

一个局部描述符表描述符的例子如图 10-15 所示。

先看访问权字节，其第 4 位为 0 则为特殊数据段描述符。由它的 TYPE 场为 0010（即 2）确定为局部描述符表描述符。最高位 P＝1，则此段在物理存储器中。其特权级 DPL＝01 即为 1。段的基地址为 010000H，段的界限为 4K。

另一类系统控制描述符为控制转移描述符，这类描述符通常称为门描述符，简称为门。与代码段描述符不同，门描述符定义了一个代码段的保护入口。门的种类有：调用门、任务门、中断门和自陷门。

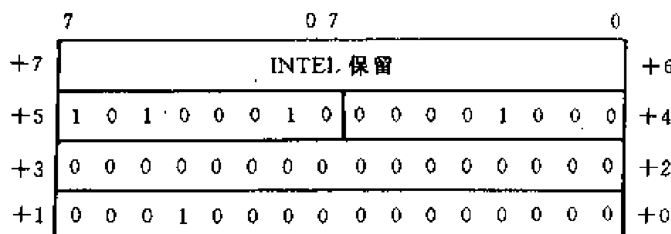
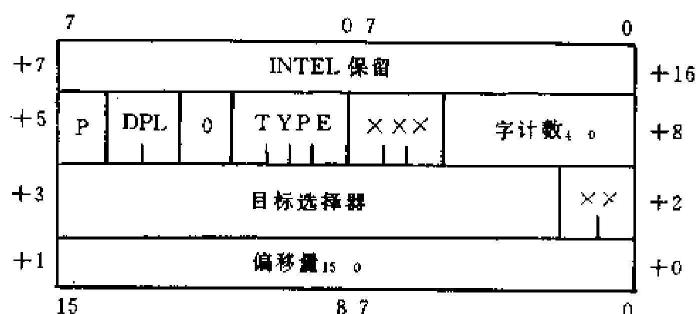


图 10-15 一个局部描述符表描述符

门是一种关卡，80286 设置门这种关卡，就在控制转移的源和目的之间，建立一道关卡。这样CPU 就可以在这道关卡上，自动地执行保护检查和控制转移到目的入口。调用门可用来改变特权层，任务门可用来实现一个任务转换，而中断门和自陷门则用来指定中断服务程序。

门描述符的数据格式和各字段的定义如图 10-16 所示。



字段名	值	说明
TYPE	4 5 6 7	调用门 任务门 中断门 自陷门
P	0 1	描述符的内容无效 描述符的内容有效
DPL	0～3	描述符特权层
字计数	0～31	要从调用程序的堆栈中复制到调用的子程序的堆栈中去的字数，这个字段只有调用门才用
目标选择器	16 位选择器	目标代码段（若是调用、中断或自陷门）的选择器 目标任务状态段（任务门）的选择器
偏移量	16 位偏移量	目标程序的段内偏移量

图 10-16 门描述符

门描述符中的访问权字节中的第 4 位为 0，P 位和 DPL 位的意义同前，类型场 TYPE 的编码确定了门的类型。由于门主要用于控制转移，故要确定目标程序的入口。目标程序的入口同样是由选择器（由选择器从描述符表中取出目标程序段的描述符）和偏移量两部分组成，故在门描述符中规定了这两部分。在调用门中还规定了字计数场，用以规定要从调用程序复制到被调用的程序中去的参量的个数。

当使用一个门时，若其中的目标选择器引用的描述符类型不正确，就会引起异常 13。

一个调用门的例子如图 10-17 所示。

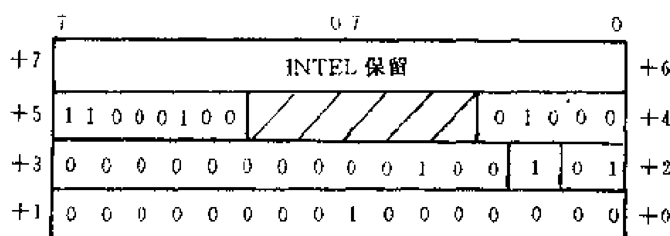


图 10-17 一个调用门描述符

其中第四位 S＝0，类型场的编码为 0100＝4 即为调用门。P＝1，表示此门的内容有效。DPL 的编码为 10 即特权级为 2。因是调用门，故字计数场有效，其值为 8，即要通过堆栈传递的参数个数为 8。目标选择器的 TI 场为 1，故要从局部描述符表的第 4 项中读取目标代码段的描述符，该选择器的特权层为 1(RPL＝1)。被调用程序的段内偏移量为 80H。

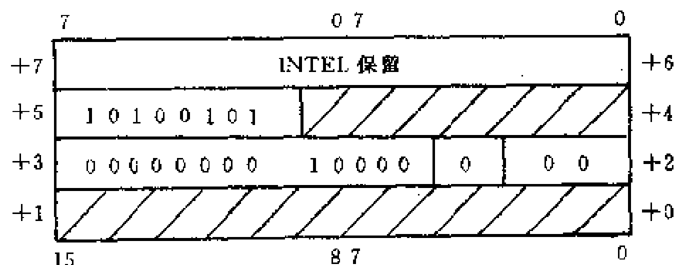


图 10-18 一个任务门描述符

一个任务门的例子如图 10-18 所示。

由访问权字节的第 4 位 S＝0，类型场的编码为 0101＝5，可以确定为一个任务门。其特权层 DPL＝01 即为特权级 1。这个门所对应的任务状态段在全局描述符表中(TI＝0)。选择器偏移量为 10H，RPL＝0。由于任务门只能引用一个任务状态段。故该任务门中的偏移量字段是不用的。

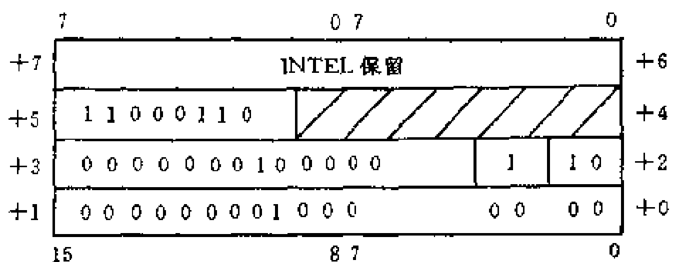


图 10-19 一个中断门描述符

一个中断门的描述符如图 10-19 所示

由访问权字节可知 S＝0，TYPE 场的编码为 0110＝6，即为一个中断门。中断服务程序的代码段的段描述符，要从局部描述符表(TI＝1)的 20H 项取出，它的段内偏移量为 80H。中断门与自陷门的功用基本上是相同的，只是引用了中断门后就自动关中断，而自陷门却不关中断。

(4) 段描述符缓冲寄存器

段描述符缓冲寄存器是 80286 内部对描述符这样的数据结构的硬件支持，是实地址方式下的段寄存器的扩展。其结构如图 10-20 所示。

每个段寄存器对应一个缓冲寄存器，每一个缓冲寄存器由访问权字节、24 位段基地址、16 位段容量构成，共 48 位。这些缓冲寄存器对于程序员是透明的。每当把一个选择器装入一个段寄存器时，这个选择器所决定的描述符就自动地装入到相应的缓冲寄存器中。以后对存储器中该段的访问，就不用由选择器从存储器的描述符表中取出相应的描述符，并由描述符中的基地址加上要访问单元的偏移量来确定最后的物理地址，而可以直接由缓冲寄存器中的段基地址与偏移量相加形成物理地址，从而缩短了访问存储器的时间。也就是说，80286 利用了描述符缓冲寄存器这样的硬件支持，使得采用描述符这样的数据结构后对内存的访问并不降低速度。80286 中并没有对这些缓冲器访问的指令，这些寄存器的内容，只有在一个段寄存器被

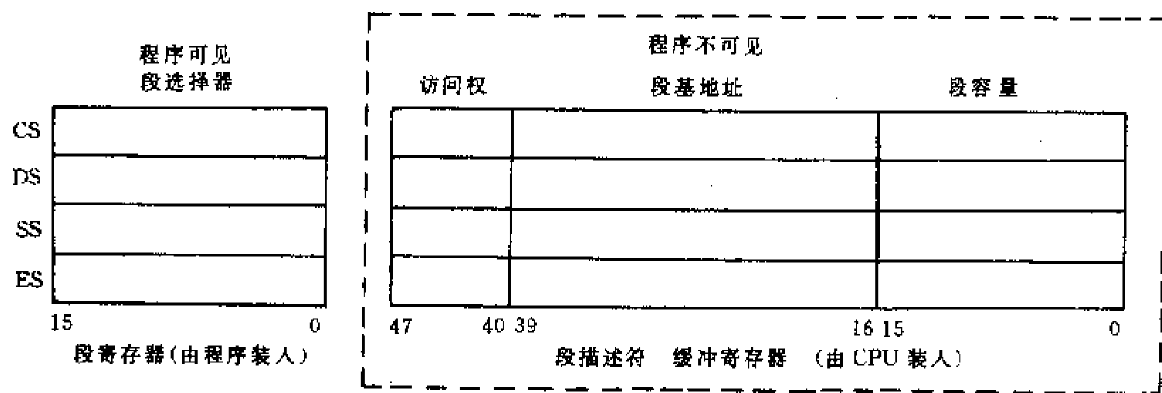


图 10-20 描述符缓冲寄存器

装入时才会改变。

用四个段寄存器进行寻址时的情况如图 10-21 所示。

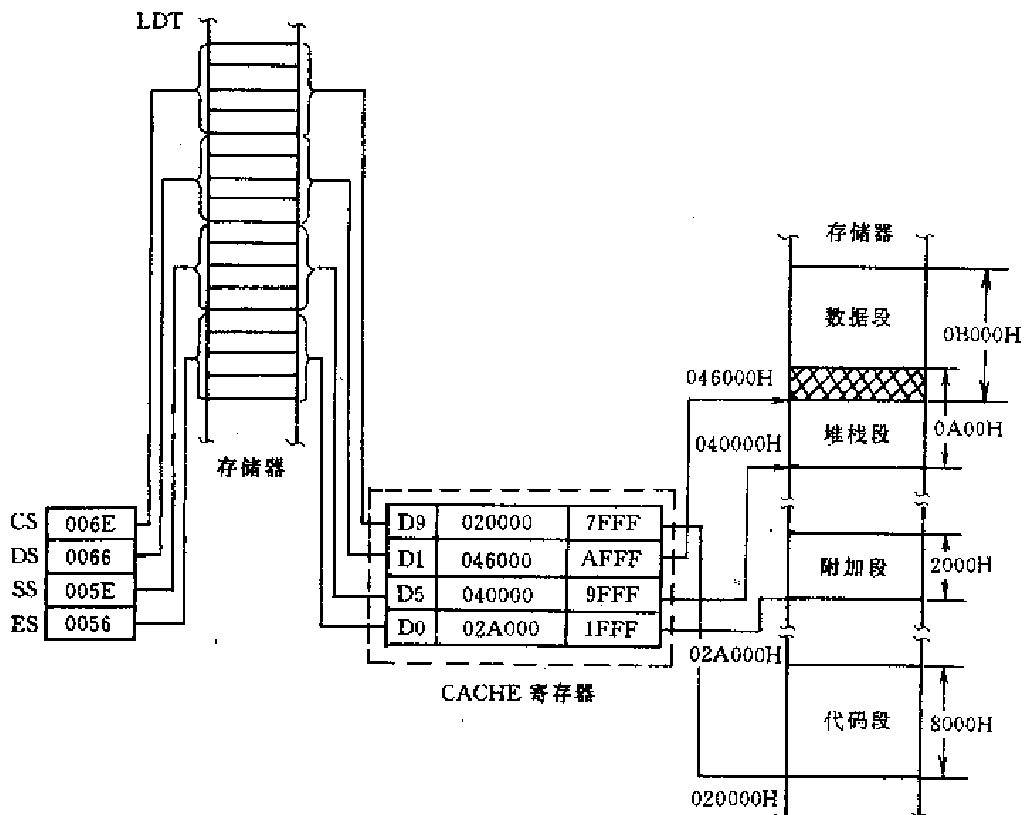


图 10-21 由缓冲寄存器确定存储器中 4 个当前段的示意图

(5)全局和局部描述符表

前面提到的各种描述符的集合形成了描述符表。系统中有三种描述符表:全局描述符表、局部描述符表和中断描述符表。

全局描述符表 GDT(Global Descriptor Table),包含着可供所有任务使用的描述符。GDT 包含除了中断和自陷门描述符之外的所有类型的描述符。

局部描述符表 LDT(Local Descriptor Table),包含着一个作业所专有的描述符。每个任务可以有它专有的 LDT。LDT 与 GDT 不同,它只能包含段、任务门和调用门描述符。

若在某一时刻，某个段的描述符在上述两个表中都不存在，则此段就不能被一个任务所访问。

每个描述符表最多可容纳 8192 个描述符。这是由于一个段寄存器中只有高 13 位用作在一个描述符表中寻址某个描述符的索引值，而 $2^{13}=8192$。

80286 的硬件中，分别有一组寄存器用来存放两个表在物理存储器中的 24 位基地址与 16 位的界限值，如图 10-22 所示。

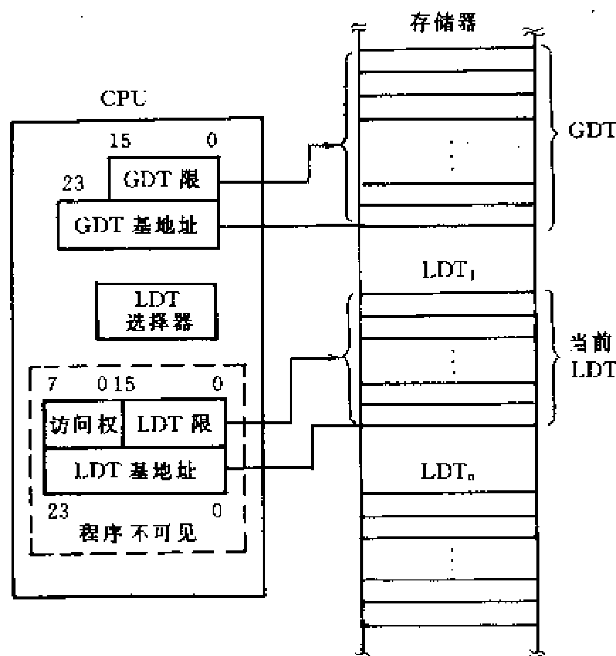


图 10-22 全局和局部描述符表定义

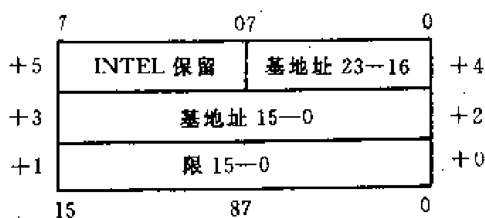


图 10-23 装入全局描述符表寄存器的数据类型

80286 的指令系统中的指令 LGDT 和 LLDT，是用来把 24 位的基地址和 16 位的限装入全局和局部描述符表 GDT 和 LDT 寄存器的。LGDT 和 LLDT 指令是保护指令，只有在特权级 0 操作的程序才能执行它们。LGDT 指令把图 10-23 中所示的 6 个字节的内容装入 GDT 寄存器。

每一个局部描述符表的 24 位物理基地址和 16 位限，必须作为一个描述符存放在全局描述符表中。故 LLDT 指令与 LGDT 指令不同，它只是把一个在 16 位寄存器中的或在一个 16 位存储器中的操作数，装入 LDT 选择器寄存器。然后用这个寄存器中的值作为选择器，从全局描述符表中取出此局部描述符表的 24 位物理基地址和 16 位限，装入 LDT 寄存器。这个装入过程与上述的描述符缓冲寄存器的装入过程一样也是自动进行的。

(6) 中断描述符表

在实地址方式下，把 256 个可能的中断处理程序的入口地址集合在一起，形成了一个中断向量表放在内存的前 1K 单元。每个中断处理程序的入口地址由一个 16 位的段基地址和一个 16 位的偏移量组成。

但在 80286 的虚地址保护方式下，段基地址是 24 位的，而且采用了描述符这样的数据结构，所以在虚地址方式下获得中断处理程序的入口地址的方法就有了变化，系统中首先要有一个中断描述符表。

中断描述符表 IDT(Interrupt Descriptor Table)最多可定义 256 种中断，此表中只能包含任务门、中断门和自陷门。中断描述符表的 24 位基地址和 16 位的限，存放在 IDT 寄存器中，如图 10-24 所示。

LIDT 指令与前述的 LGDT 指令类似，是把 6 个字节的值(包括 3 个字节的段基地址和 2 个字节的限)装入 IDT 寄存器。

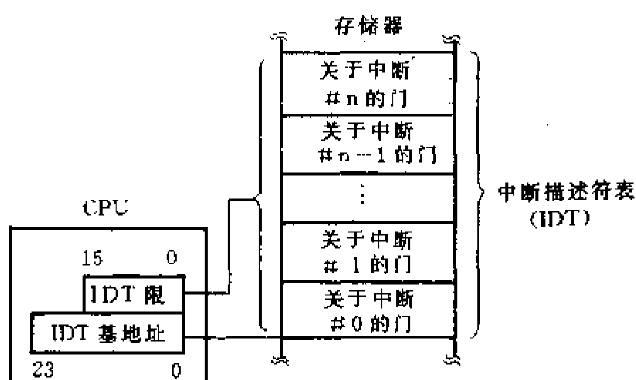


图 10-24 中断描述符表定义

寻找IDT表中各项的操作，是由INT指令、外部中断或异常来进行的。由于INTEL公司保留了32种中断，所以IDT必须至少要有256个字节来为所有保留的中断分配空间(32×8=256)。在保护方式下，80286寻找中断处理程序入口地址的方式与实地址方式下不同。当接收到一个中断向量后，就自动地引用中断描述符表寄存器。通过是否越限检查之后，就用中断描述符表基地址寄存器中的值和中断向量(相当于变址值)，共同确定该中断所对应的中断门在表中的位置。通过访问权检查后，就引用该中断门。利用该中断门中的选择器，到全局描述表中寻找中断处理程序所在代码段的描述符。找到以后，经过访问权检查，引用该代码段描述符，就找到了该代码段的24位基地址。与中断门中给定的16位偏移量相加，就找到了该中断处理程序的入口地址，从而能开始执行中断服务。

如上所述，为了支持虚地址保护方式的寻址，80286提供了三种描述符表如图10-25所示。

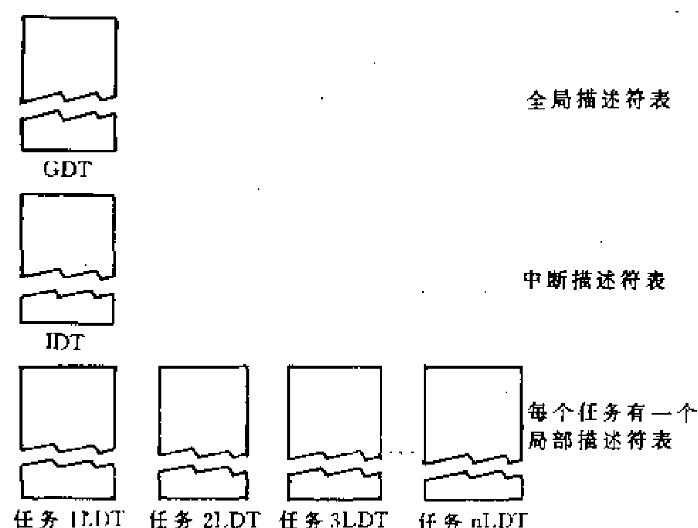


图 10-25 80286中的描述符表

如上所述，80286中是用选择器和描述符的数据结构来实现在虚地址保护方式下的寻址的。这样做的目的，不仅是为了能寻址16兆的实存地址空间，更重要的是给操作系统和80286的虚地址保护机构，提供了一个广阔的活动舞台。为了避免因引用这些数据结构，而频繁地访问存储器，80286提供了相应的硬件寄存器，从而使因访问这些数据结构而产生的对处理器运行速度的影响减至最小。这些硬件寄存器如图10-26所示。

下面举一个例子来看80286在虚地址保护方式下是如何寻址的。

若要寻址一个操作数，虚地址指示器由两部分组成，一部分由DS中的内容规定，另一部分是段内偏移量。要找到段基地址，就要以DS中的内容作为索引，从描述符表中取出相应的

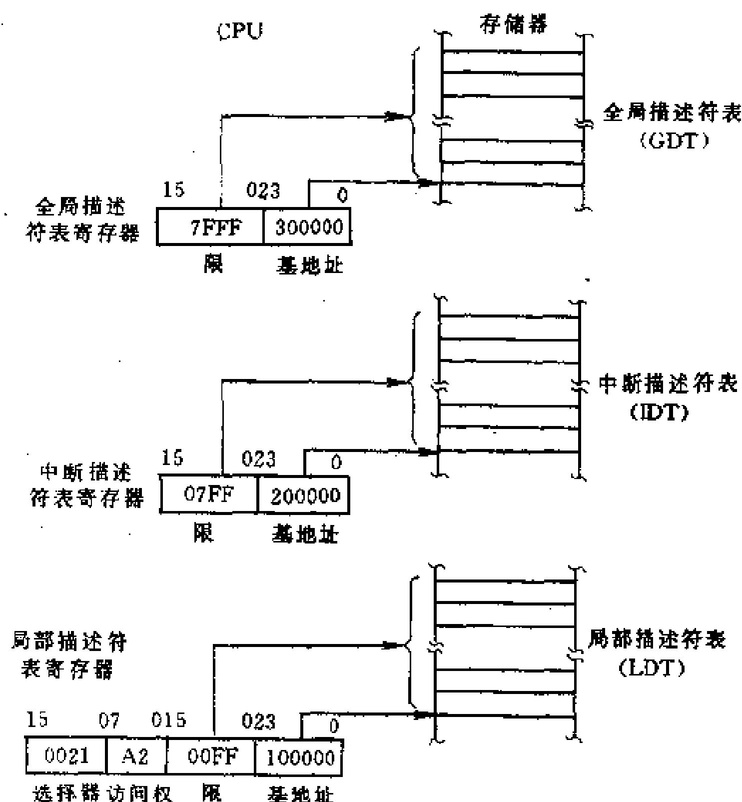
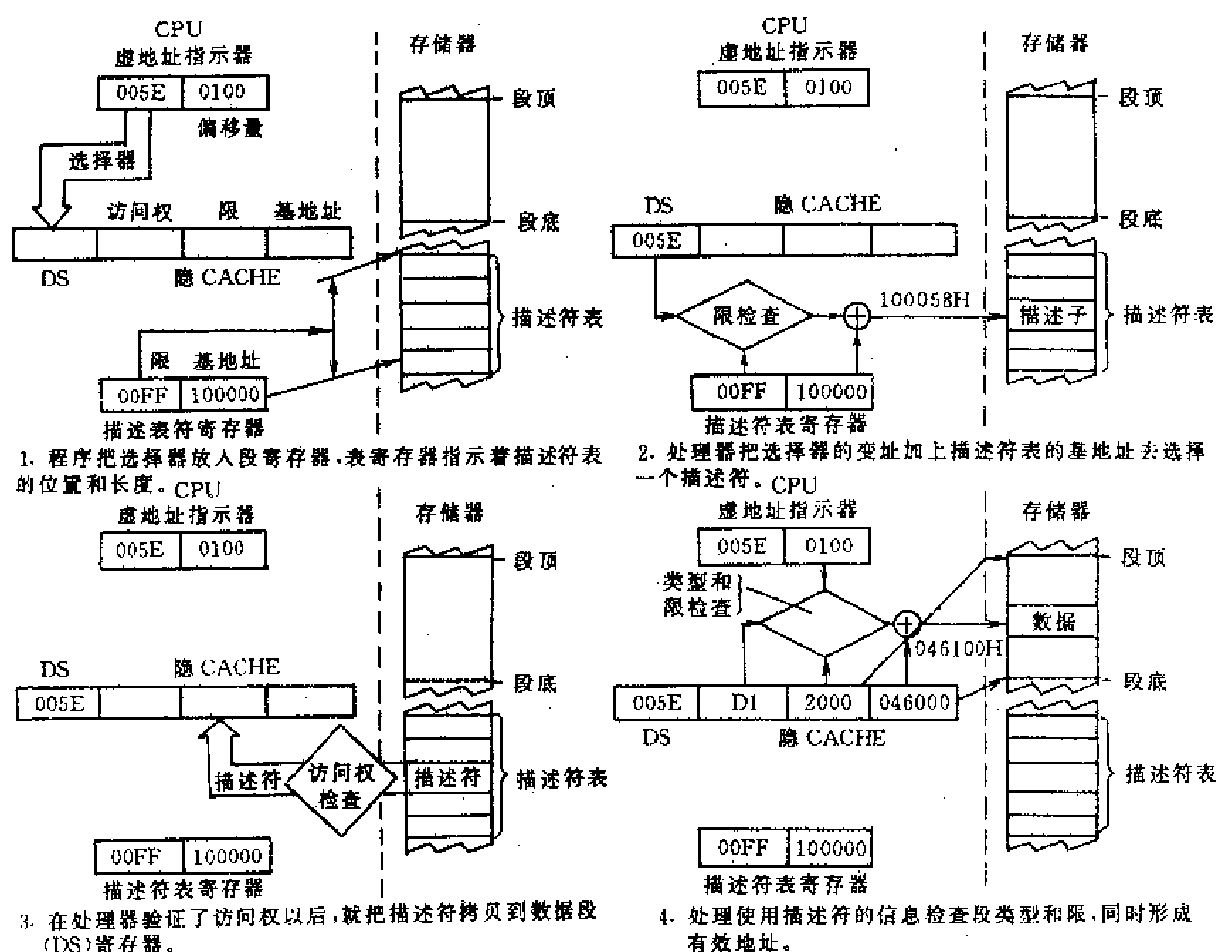


图 10-26 描述符表寄存器和描述符表

描述符。而此描述符表在内存中的地址由描述符表寄存器规定，如图 10-27 所示。

图中指示描述符表的基地址为 100000H，其限为 00FFH。DS 中的内容为 005EH，故小于限，于是由 100000H 加上 005EH，就可以找到相应的描述符。在处理器验证了访问权以后，就把描述符中的段基地址和界限值，送至 CPU 中的描述符寄存器中。接着处理器对段的类型和限进行检查后，就以描述符寄存器中的段基地址加上操作数虚地址指示器中给定的偏移量，就可以最后确定所寻址的操作数的物理地址。



1. 程序把选择器放入段寄存器，表寄存器指示着描述符表的位置和长度。

2. 处理器把选择器的变址加上描述符表的基地址去选择一个描述符。

3. 在处理器验证了访问权以后，就把描述符拷贝到数据段(DS)寄存器。

4. 处理使用描述符的信息检查段类型和限，同时形成有效地址。

图 10-27 寻址一个操作数的例子

第二节 80386 微处理器

一、概述

1985 年 10 月 Intel 公司推出了它的与 8086、80286 相兼容的高性能的 32 位微处理器 80386。80386 是能满足高性能的应用领域与多用户、多任务操作系统需要设计的先进的 32 位微处理器。它的主要性能如下：

1. 灵活的 32 位微处理器：

■8 位、16 位或 32 位数据类型；

■8 个通用的 32 位寄存器。

2. 较大的寻址空间：

■4 千兆字节物理空间；

■64 兆兆字节虚拟空间；

■存储器的分段结构，一个段最大可达 4 千兆字节。

3. 具有集成的存储器管理部件：

■支持虚拟存储器；

■可选择的片内分页机构；

■4 级保护；

■与 80286 完全兼容；

4. 目标码与所有 8086 系列的微处理器兼容；

5. 虚拟的 8086 方式允许在受保护的和分页的系统中运行 8086 的软件；

6. 支持硬件调试。

7. 优化的系统性能：

■指令的流水线结构；

■具有片内地址转换的高速缓冲存储器；

■时钟为 12.5MHz 和 16MHz；

■总线带宽为 32 兆字节/秒。

8. 通过 80287 和 80387 协处理器支持高速数值处理。

9. 完整的系统开发支持工具：

■软件：C・P/LM，汇编语言系统生成工具；

■调试器：PSCOPE、ICE™——386。

10. 高速 CHMOS Ⅲ 技术；

11. 132 引脚网络阵列式封装；

80386 有两种档次的芯片：80386SX 和 80386DX。80386SX 是准 32 位芯片，即其内部结构为 32 位，而外部数据线只有 16 条，80386DX 是真正的 32 位芯片，内部、外部都是 32 位。80386SX 只能配接协处理器 80287，而 80386DX 需配接协处理器 80387。

二、80386 的基本结构

1. 80386 的宏结构

80386 的宏结构如图 10-28 所示。它是由中央处理器(CPU)、存储器管理部件(MMU)和总线接口部件(BIU)组成的。

图 10-28 80386 的宏结构

中央处理器又是由指令部件和执行部件所组成的。指令部件可以预取指令，且对指令操作码进行译码，并把它们存放在已译码的指令队列里供执行部件使用（省去取指令和译码的时间）。执行部件包括 8 个既可以用于数据操作，也可以用于地址计算的 32 位通用寄存器。还包括一个 64 位的桶形移位器(barrel shifter)，用于加速移位、循环以及乘除法操作，这使典型的 32 位乘法可在 1 微秒内执行。

存储器管理部件(MMU)由分段部件和分页机构组成。分段部件通过提供一个额外的寻址器件对逻辑地址空间进行管理，可以实现任务之间的隔离也可以实现指令和数据区的再定位。分页机构提供了对物理地址空间的管理，每一页 4K 字节；每一段可以是一页也可以是若干页。为了实现虚拟存储器系统，80386 对所有的页和故障都支持完整的再启动功能。

存储器是按照段来组织的，每一段的大小都可以达到 4 千兆字节。一个给定范围的线性地址空间或一个段可以有相应的属性。这些属性包括它的位置、大小、类型(即是堆栈、码或数据)及其保护特性。80386 的每一个任务都可以有最多 16381 个段，每段最大都可达 4 千兆字节。因此，80386 为每一个任务都可提供最大为 64 兆兆字节的虚拟存储器。

为了使应用程序和操作系统相互隔离和各自得到保护，分段部件提供了 4 级保护。这种由硬件实施的保护，使各种系统的设计具有高度的完整性。

80386 有两种操作方式：实地址方式（Real Address Mode）和受保护的虚拟地址方式(Protected Virtual Address Mode)。在实地址方式下，80386 的操作像一个极快的 8086，不同的是如果需要可以扩展到 32 位。保护方式提供了对复杂的存储器管理部件的存取，以及分页和处理器的各种特殊性能。

在保护方式下，软件可以通过切换进入虚拟的 8086 方式。在这种方式下的每个任务都用 8086 的语义运行，从而可以运行 8086 的各种软件(应用程序或整个操作系统)。通过采用分页和模拟 I/O 指令，各种虚拟的 8086 任务可以与 80386 主操作系统相互隔离并受到保护。

2. 寄存器结构

80386 共有七类 32 个寄存器，它们是：通用寄存器，段寄存器，指令指针和标志寄存器，控制寄存器，系统地址寄存器，排错寄存器，测试寄存器。

这些寄存器包括了全部的 16 位的 8086，80186 和 80286 的寄存器。这些前三类寄存器如图 10-29 所示。

其它类型的寄存器，如控制、系统地址、调试和测试寄存器，主要是用于简化设计和对操作系统进行调试。

(1) 通用寄存器

80386 中有 8 个 32 位的通用寄存器如图 10-30 所示。

每一个寄存器都可以存放数据或地址量，它们支持 1、8、16、32 和 64 位的数据操作数以及 1 到 32 位的位场操作数。它们还支持 16 位和 32 位的地址操作数。这些通用寄存器是 8086、80286 的 16 位通用寄存器的扩展，故命名为 EAX、EBX、ECX、EDX、ESI、EDI、EBP 和 ESP。

对这些寄存器的低 16 位可以分别存取，好像 8086 中的 16 位寄存器一样使用，而且它们的命名也与 8086 中的相同。

数据寄存器 EAX、EBX、ECX、EDX 中的最低字节(位 0～7)和较高字节(位 8～15)可以作为 8 位的寄存器单独存取。它们的命名也与 8086 的一样，如图 10-30 所示。

(2) 指令指针和标志寄存器

80386 的地址线是 32 条，故指令指针为 32 位寄存器，是 IP 的扩展故称为 EIP。EIP 中存

通用数据和地址寄存器

31　　16	15　　0	
	AX	EAX
	BX	EBX
	CX	ECX
	DX	EDX
	SI	ESI
	DI	EDI
	BP	EBP
	SP	ESP

段选择器寄存器

15　　0		
	CS	码
	SS	堆栈
	DS	数据
	ES	数据
	FS	数据
	GS	数据

指令指针和标志寄存器

31　　16	15　　0	
	IP	EIP
	FLAGS	EFLAGS

图 10-29　80386 中的基本结构寄存器

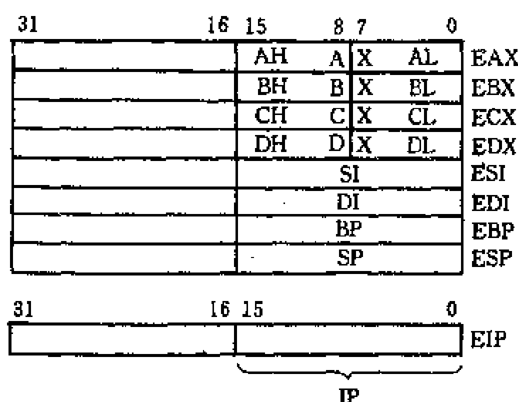


图 10-30　通用寄存器和指令指针

放的始终是下一条要取出的指令的偏移量，此偏移量是相对于码段(CS)基址的偏移量。EIP 的低 16 位(位 0～15)称为 IP，它由 16 位的地址操作数使用。

80386 中的标志寄存器是一个命名为 EFLAGS 的 32 位寄存器，它是由 80286 的标志位扩展而成，所定义的位和位场如图 10-31 所示。它的低 16 位(位 0～15)包含了命名为 FLAGS 的 16 位的标志寄存器，即为 80286 中的标志寄存器，它的低 12 位(位 0～11)即为 8086 的标志寄存器。所以，在执行 8086 和 80286 操作码时，这个 16 位标志寄存器是最有用的。

标志

EFLAGS：为 Intel 公司保留 (位 31～18)，VM (17)，RF (16)，0 (15)，NT (14)，IOPL (13、12)，OF (11)，DF (10)，IF (9)，TF (8)，SF (7)，ZF (6)，0 (5)，AF (4)，0 (3)，PF (2)，1 (1)，CF (0)

虚拟方式　恢复方式　嵌套任务标志　I/O 特权级　溢出　方向标志　中断允许

进位标志　奇偶标志　辅助进位　零标志　符号标志　陷井标志



图 10-31　80386 中的标志寄存器

80386 扩展的标志位为 VM(位 17)和 RF(位 16)下面主要介绍这两个标志位的功能。

VM(Virtual 8086 Mode：虚拟 8086 方式，位 17)

80386 的保护方式中，由 VM 位提供一种虚拟 8086 方式。即当 80386 处于保护方式时，如果使 VM 位置位，80386 将转为虚拟 8086 方式。VM 位只能在保护方式下由 IRET 指令(若当前的特权级＝0)或在任何特权级下由任务切换设置。VM 位不受 POPF 指令的影响，PUSHF 指令总是使该位清零。在中断处理过程中被压入或在任务切换期间被保存的

EFLAGS 的映象中的 VM 位将包含一个 1，条件是被中断的码正作为虚拟的 8086 任务而被执行。

RF 标志(Resume Flag：恢复标志，位 16)是与调试寄存器的断点或单步操作一起使用的。在断点处理之前，在两条指令之间对该位进行检查。RF 位置位时，则在下一条指令执行期间不顾任何调试故障。然而，每当成功地完成一条指令(表明没有故障)时，RF 标志都将自动复位。但是执行 IRET 指令、POP 指令和引起任务切换的 JMP、CALL 和 INT 指令时例外。这些指令设置 RF 的值为根据存储器映像所确定的值。例如，在断点服务子程序的结尾处，IRET 指令可以弹出一个带有 RF 位置位的 EFLAG 映像，并在断点地址处恢复程序的执行而不致在同一位置上产生另一次断点故障。

(3)段寄存器

在 80386 中存储单元的地址仍是由两部分组成，即段基地址与段内偏移量。只是在 80386 中，段内偏移量是 32 位的，可由各种寻址方式(在后面介绍)确定；段的基地址也是 32 位的，但是它不是由段寄存器中的值直接确定的，而是与 80286 中一样保存在一个表中，段寄存器的值只是表的索引。

在 80386 中有 6 个 16 位段寄存器也称为选择器，即 CS、SS、DS、ES、FS 和 GS。

CS 选择器表示当前的码段；SS 选择器表示当前的堆栈码；而 DS、ES、FS 和 GS 都可以用来表示当前的数据段。

在 80386 中每一个段寄存器(此寄存器是程序员可见的)，都有一个与之相联系的但程序员不可见的段描述符(用以描述一个段，例如此段的基地址，段的大小和段的属性等)寄存器，如图 10-32 所示。每个段描述符寄存器都保存着一个 32 位的段基地址，一个 32 位的段界限(大小)和其他一些必要的属性。

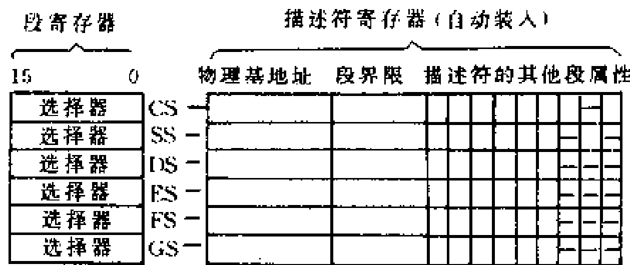


图 10-32 段描述符寄存器

每当一个段寄存器中的值确定以后，80386 中的硬件会自动根据段寄存器中的值即索引，从相应表中取出一个 8 字节描述符，装入到相应的段描述符寄存器中。每当出现存储器访问时，就可由所用的段寄存器，直接用相应的段描述符寄存器中的段基地址作为线性地址计算中的一个元素，而不必在访问时去查表，查到段基地址。这就是 80386 对它的寻址方式的硬件支持，从而加快了存储器访问。

(4) 系统地址寄存器

在 80286 和 80386 中，是利用选择子(选择器)和描述符这样的数据结构来确定存储单元的段地址的，这样做不仅可以用只有 16 位的段寄存器(段选择子)来确定 24 位(80286)或 32 位(80386)的段基地址，更重要的是可以确定段的一些属性例如特权，从而为 80286、80386 的操作系统的保护机构提供了广阔的活动舞台。

80286 和 80386 中的段基地址是由一个 8 个字节的描述符(如图 10-33 所示)所确定的。

由相关的述符组成一个表，这些表是：

GDT(Global Descriptor Table：全局描述符表)



基址—段的基地址
界限—段的长度
P—存在位 1=存在 0=不存在
DPL—描述符特权级 0—3
S—段描述符 0=系统描述符 1=码或数据段描述符
类型—段的类型
A—已存取位
G—粒度位 1=段长度为页粒度 0=段长度为字节粒度
D—缺省操作数的大小(仅在码段述符中识别) 1=32 位段 0=16 位段
0—为与将来的处理器兼容必须设置为零(0)的位

图 10-33 80386 中的段描述符

IDT(Interrupt Descriptor Table:中断描述符表)

LDT(Local Descriptor Table;局部描述符表)

TSS(Task State Segment:任务状态段)

这些表的基地址和它们的界限(大小)由相应的寄存器保存,如图 10-34 所示。

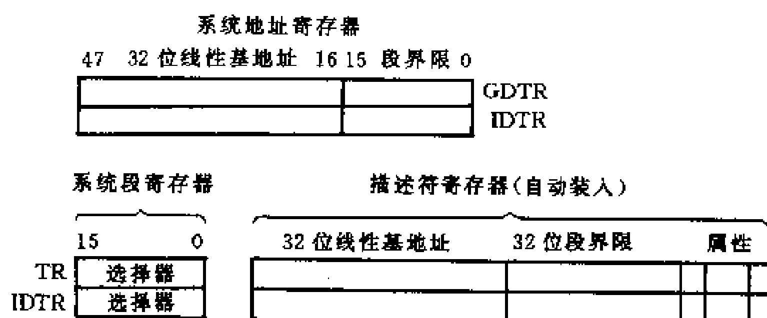


图 10-34 系统地址和系统段寄存器

GDTR 和 IDTR

这两个寄存器分别保存着 GDT 和 IDT 的 32 位的线性基地址以及 16 位的界限值。全局描述符表 GDT 和中断描述符表 IDT 的界限都是 16 位的,即每一个表最大是 64K,每个描述符为 8 个字节,故每个表最大可以有 8K 个描述符。实际上在 80286 和 80386 中,最多只有 256 个中断或异常向量,故 IDT 表中最多只有 256 个中断描述符。

由于 GDT 和 IDT 对系统中的所有任务都是全局性的,因此,GDT 和 IDT 所在的段由 32 位的线性地址(如果允许分页则指向页转换)和 16 位的界限值确定。

LDTR 和 TR

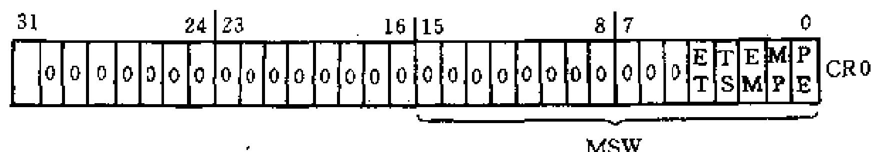
局部描述符表 LDT 和任务状态段 TSS 是面向任务的,故它们所在的段不是由这些表本身决定的,而是由任务决定的,即由任务的系统段寄存器中的选择器值决定的。故局部描述符表寄存器 LDTR 和任务状态段寄存器 TR 中的值,只是一个 16 位的选择器。如图 10-34 中所示。但 80386 中提供了相应的自动装入的、不可见的描述符寄存器,作为对存储单元访问的硬件支持以加快存储单元的访问。

(5)控制寄存器

80386 中有 3 个 32 位的控制寄存器 CR0、CR2 和 CR3，以保存全局性(不是特定的个别任务)的机器状态。这些寄存器与上面介绍的系统地址寄存器一起，保存着影响系统中所有任务的机器状态。

a. CR0：机器控制寄存器(包括 80286 的机器状态字)

CR0 中定义了 6 个控制和状态位，如图 10-35 所示。



注：0表示 Intel 公司已保留，不要定义。

图 10-35 控制寄存器

注：0 表示 Intel 公司已保留，不要定义。

为了与 80286 的保护方式兼容，CR0 的低 16 位还作为机器状态字(Machine Status Word)MSW。

CR0 中各位的定义如下：

PG(Paging Enable：分页允许，第 31 位)

若 PG 位置位，允许片内分页部件工作；否则，禁止分页部件工作。

ET(Processor Extension Type：处理器扩展类型，第 4 位)

ET 表示所扩展的协处理器的类型，即为 80287 或 80387。这可由 80386 复位后，由 ERROR# 输入信号的电平来确定。如果需要，在程序控制之下，通过对 CR0 送数，可以使 ET 位置位或复位。如果 ET 位置位，就使用与 80387 兼容 的 32 位规程；如果 ET 位复位，则采用与 80287 兼容的 16 位规程。

注意：为严格保持与 80286 的兼容性，ET 位并不受 LMSW 指令的影响。但当存储 MSW 或 CR0 时，位 4 准确地反映 ET 位的当前状态。

TS(Task Switched：任务切换，第 3 位)

不论什么时候完成任务切换操作，TS 位都自动置位。一条协处理器操作码如果使 TS 位置位且 MP 位也置位，将引起协处理器无效陷井(异常 7)。此陷井处理程序，典型地保存属于上一个任务的 80287/80387 内容，装载属于当前任务的 80287/80387 状态，且在返回失败的协处理器操作码之前，清除 TS 位。

EM(Emulate Coprocessor：模拟协处理器，第 2 位)

若 EM 位置位，将使所有的处理器操作码都产生协处理器无效陷井(异常 7)。若 EM 位复位，将允许协处理器操作码在实际的 80287 或 80387 上执行(在复位后默认状态认为是 80387)。

注意：WAIT 操作码不受 EM 位置位的影响。

MP(Monitor Coprocessor：监控协处理器，第 1 位)

MP 位与 TS 一起使用以确定在 TS=1 时 WAIT 操作码是否产生协处理器无效陷井(异常 7)。即当 MP=1 且 TS=1 时，WAIT 操作码产生陷井；否则，WAIT 操作码不产生陷井。

注意：当任务切换操作完成时，TS 是自动置位的。

PE(Protection Enable：保护允许，第 0 位)

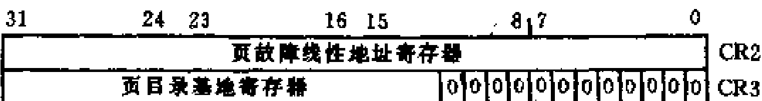
要使 CPU 进入保护方式，必须使位置位，若 PE 位复位，CPU 再次处于实模式。PE 位可以

由加载 MSW 或加载 CR0 指令置位;但为了严格与 80286 兼容,PE 位不能由 LMSW 指令复位。

(CR1:保留)

b. CR2:页故障线性地址

CR2 保存了一个 32 位的线性地址。如图 10-36 所示。



注:0表示 Intel 公司已保留,不要定义。

图 10-36 控制寄存器 2 和 3

该地址是在检测出的最后的页故障所产生的。当它被引用时,被推至页失败处理程序堆栈上的错误码,将提供此页失败的附加的状态信息。

c. CR3:页目录基地址

CR3 如图 10-36 所示。包含着页目录表的物理基地址。80386 的页目录表总是按页对齐的(4K 字节对齐)。因此,当写 CR3 时,它的低 12 位是忽略的,当存 CR3 时,它们也是未定义的。

若通过 TSS 改变在 CR3 中的值的任务切换,或明显地用任何值装载 CR3。将使在分页单元高速缓冲器中的所有缓存的页表输入项无效。

在 80286 中为了存取 MSW,有装载(load)MSW 的指令 LMSW,和存储(store)MSM 指令 SMSW 。在 80386 中也有相应的对 CR0 操作指令 LMSW 和 SMSW。但是为了与 80286 操作系统兼容, 80386 的 LMSW 指令以和 80286 的 LMSW 指令相同的形式工作,即仅对 CR0 的低 16 位操作,而不管 CR0 上新增加的位。为了对整个 32 位的 CR0 操作,应使用指令 MOV CR0 ,Reg。

(6) 排错(Debug)寄存器

80386 中有 6 个程序员可存取的排错寄存器,如图 10-37 所示。

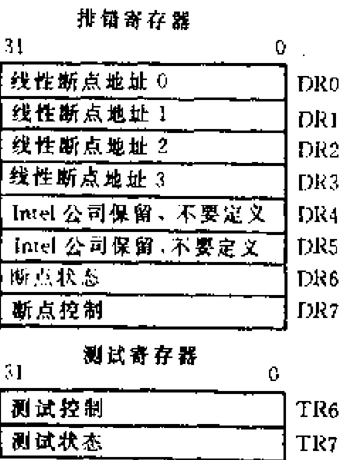


图 10-37 排错和测试寄存器

这 6 个寄存器用于排错,寄存器 DR0~DR3 可指定 4 个线性断点地址,寄存器 DR6 用于设置断点,而寄存器 DR7 用于显示断点的当前状态。

(7)测试(Test)寄存器

80386 有两个测试寄存器 TR6 和 TR7。

以上寄存器的可存取性与兼容性如下:

在真实方式和保护方式中,有关寄存器的可存取性存在着一些差别。表 10-7 归纳了这些差别。

80386 中的寄存器的某些位是没有定义的,当调出无定义位时,要按照完全无定义对待 。为使用户软件与将来的处理器兼容,请遵循下列准则:

①. 当测试有定义的寄存器位之值时,不要依据任何无定义位的状态. 测试时将它们屏蔽掉。

表 10-7　寄存器的使用

寄　存　器	用于真实方式		用于保护方式		用于虚拟方式	
	装　入	存　储	装　入	存　储	装　入	存　储
通用寄存器	是	是	是	是	是	是
段寄存器	是	是	是	是	是	是
标志寄存器	是	是	是	是	IOPL	IOPL * *
控制寄存器	是	是	PL=0	PL=0	否	是
全局描述符表寄存器	是	是	PL=0	是	否	是
中断描述符表寄存器	是	是	PL=0	是	否	是
局部描述符表寄存器	否	否	PL=0	是	否	否
任务寄存器	否	否	PL=0	是	否	否
排错寄存器	是	是	PL=0	PL=0	否	否
测试寄存器	是	PL=0	PL=0	PL=0	否	否

注：PL=0：只有当现行特权级别为零时，才可以对该寄存器进行存取。

* * IOPL：PUSH 和 POPF 指令造成虚拟 8086 方式的 I/O 特权级敏感

②. 在把寄存器的内容存到存储器或另一个寄存器时，不要依据任何无定义位的状态。

③. 不要以写入任何无定义位的某些信息作为依据。

④. 装载寄存器时，总是用零装入无定义位。

⑤. 已经预先存储的寄存器可能不带屏蔽被重新装入。

依赖无定义位的值将使用户软件依赖于 80386 对这些位的不确定处理。因为将来的处理器有可能采用 80386 寄存器中的无定义位，这样用户的软件就会不兼容。故切记避免任何软件以 80386 寄存器中无定义的位的状态为依据。

三、80386 的引脚

80386 的引出脚从元件的顶面看上去如图 10-38 所示。V_{CC}(电源)和 GND(地)的连接必须形成多个 V_{CC}和 GND 引出脚。每个 V_{CC}和 GND 引出脚必须连接到相应的电压电平上。从外部靠近封装处把所有的 V_{CC}引出脚固定在一起，并类似地固定所有的 GND 引出脚。为了电源散热，电路板应当包括 V_{CC}和 GND 的平面。

引出脚的功能分组如表 10-8 所示。

80386 引脚信号：

(1)CLK_2：两倍时钟信号，输入；该信号与 82384 时钟信号同步输入 25MHz，2 分频后为 12.5MHz 或 16MHz，它为 386 的主频信号。

(2)D_0～D_{31}：数据总线，双向三态，一次可传送 8、16、32 位数据。

(3)A_2～A_{31}：地址总线，输出三态和$\overline{BE_0}$～$\overline{BE_3}$相组合可起到 32 位地址作用。

(4)$\overline{BE_0}$～$\overline{BE_3}$：字节选通输出线，每根线控制选通一个字节；$\overline{BE_0}$选通 D_0～D_7 的字节；$\overline{BE_1}$选通 D_8～D_{15}的字节，...... 相当存储器可分 4 个库，A_2～A_{31}可寻址 2^{30}；若 4 根选通线有效，则可寻址 $2^{30}\times 2^2=2^{32}$。

(5)$W/\overline{R}$：读/写控制，输出信号。

(6)$D/\overline{C}$：数/控信号，输出，表示是数据传送周期还是控制周期。

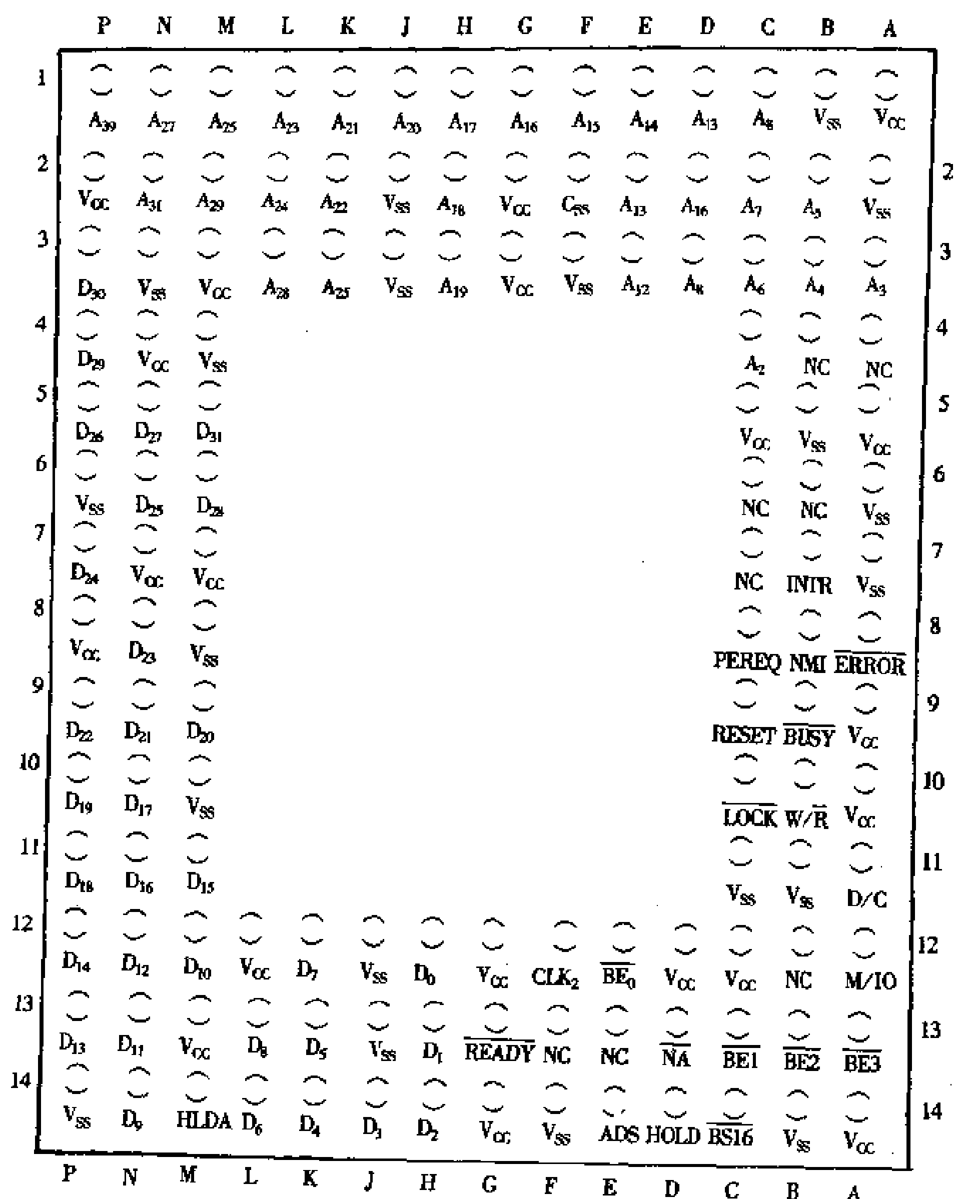


图 10-38 80386 PGA 引出脚仰视图

注：标为"N. C"的引出脚应当不连接。

表 10-8 80386 PGA 引出脚一功能分组表

引出脚/信号		引出脚/信号		引出脚/信号		引出脚/信号	
N2	A31	M5	D31	A1	Vcc	A2	V_{ss}
P1	A30	P3	D30	A5	Vcc	A6	V_{ss}
M2	A29	P4	D29	A7	Vcc	A9	V_{ss}
L3	A28	M6	D28	A10	Vcc	B1	V_{ss}
N1	A27	N5	D27	A14	Vcc	B5	V_{ss}
M1	A26	P5	D26	C5	Vcc	B11	V_{ss}
K3	A25	N6	D25	C12	Vcc	B14 V_{ss}	
L2	A24	P7	D24	D12	Vcc	C11	V_{ss}
L1	A23	N8	D23	G2	Vcc	F2	V_{ss}

（续）

引出脚/信号		引出脚/信号		引出脚/信号		引出脚/信号	
K2	A22	P9	D22	G3	Vcc	F3	V_{ss}
K1	A21	N9	D21	G12	Vcc	F14	V_{ss}
J1	A20	M9	D20	G14	Vcc	J2	V_{ss}
H3	A19	P10	D19	L12	Vcc	J3	V_{ss}
H2	A18	P11	D18	M3	Vcc	J12	V_{ss}
H1	A17	N10	D17	M7	Vcc	J13	V_{ss}
G1	A16	N11	D16	M13	Vcc	M4	V_{ss}
F1	A15	M11	D15	N4	Vcc	M8	V_{ss}
E1	A14	P12	D14	N7	Vcc	M10	V_{ss}
E2	A13	P13	D13	P2	Vcc	N3	V_{ss}
E3	A12	N12	D12	P8	Vcc	P6	V_{ss}
D1	A11	N13	D11				
D2	A10	M12	D10				
D3	A9	N14	D9	F12	CLK2	A4	N.C.
C1	A8	L13	D8			D4	N.C.
C2	A7	K12	D7	E14	$\overline{ADS}$	B6	N.C.
C3	A6	L14	D6			B12	N.C.
B2	A5	K13	D5	B10	$W/\overline{R}$	C6	N.C.
B3	A4	K14	D4	A11	D/C	C7	N.C.
A3	A3	J14	D3	A12	$M/\overline{IO}$	E13	N.C.
C4	A2	H14	D2	C10	$\overline{LOCK}$	F13	N.C.
A13	$\overline{BE3}$	H13	D1				
B13	$\overline{BE2}$	H12	D0	D13	$\overline{NA}$	C8	PEREQ
C13	$\overline{BE1}$			C14	$\overline{BS16}$	B9	$\overline{BUSY}$
E12	$\overline{BE0}$			G13	$\overline{READY}$	A8	$\overline{ERROR}$
		D14	HOLD				
C9	RESET	M14	HLDA	B7	INTR	B8	NMI

(7)$M/\overline{IO}$：存储器/I/O 选择信号，为输出信号。

(8)$\overline{LOCK}$：总线锁定，输出。

(9)$\overline{ADS}$：地址状态，三态输出信号，表示总线周期中地址信号有效。

(10)$\overline{NA}$：下一地址请求，输入信号，允许地址流水线操作，即当前周期发下一总线周期地址的状态信号。

(11)$\overline{BS16}$：总线宽度为 16 的输入信号。

(12)READY：准备就绪，输入信号，表示当前总线周期已完成。

(13)HOLD：总线保持请求，输入信号。

(14)$\overline{HLDA}$：总线保持响应，输出信号。

(15)PEREQ:协处理器请求,输入信号,表示 387 要求 386 控制它们与存储器之间的信息传送。

(16)$\overline{BUSY}$:协处理器忙,输入信号。

(17)$\overline{ERROR}$:协处理器出错,输入信号。

(18)$\overline{INTR}$:可屏蔽中断请求信号。

(19)NMI:非屏蔽中断请求信号。

(20)RESET:复位信号。

四、80386 的工作方式

1. 80386 存储器实地址方式

当 80386 复位或接通电源时,就处于实地址方式。80386 所有指令在实地址方式下都是有效的,不过操作数默认长度是 16 位,物理地址形成同 8086 一样,将段寄存器内容左移 4 位与有效地址相加而得到的,寻址空间为 1MB,只有地址线 $A_2 \sim A_{19}$,$BE_0 \sim BE_3$ 是有效的。所有的段最大为 64KB。在实地址方式下保留了两个固定的存储区域:

(1)系统初始化区:FFFFFFF0H~FFFFFFFFH

(2)中断向量表区:00000~003FFH 在 1K 存储空间中保留了 256 个中断服务程序的入口地址,每个入口地址用到 4 个字节。

2 80386 存储器虚地址保护方式

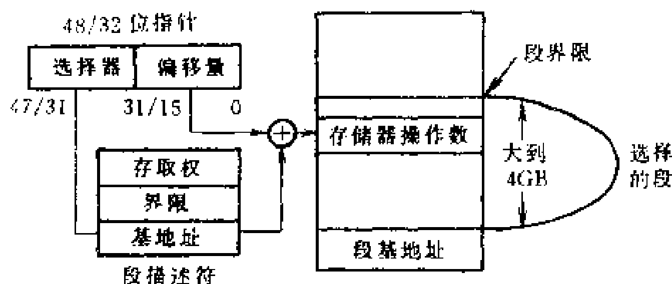


图 10-39 保护方式寻址

当 80386 在虚拟地址保护方式工作时,80386 才能尽其所长,其线性地址空间增加到 2^{32} = 4000MB,虚拟地址空间为 64MMB。而这么大的虚拟地址空间是怎样形成的呢?它和 80286 一样采用了描述符表这样的数据结构办法而形成的。

(1)存储器寻址

在保护方式下,段基地址是 32 位的,但 80386 中的段寄存器是十六位(为了与 8086 兼容)。故在保护方式下,与实地址方式不同,某个段寄存器中的内容不是段的基地址。32 位的段基地址存放在一个表(段描述符表)中,段寄存器的内容作为选择器,即用作为表的索引,可以从表中取出相应的段描述符(包括 32 位段基地址、界限和存取权等,详见下面讨论)。从表中取出的 32 位段基地址与 32 位的有效地址相加,在没有分页结构时就形成了 32 位的物理地址,如图 10-39 所示。

(2)描述符

a. 段选择器

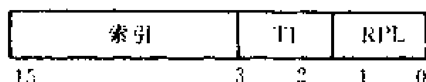


图 10-40 选择器格式

在保护方式下,段寄存器是一个选择器和 80286 一样,它有 3 个字段组成,如图 10- 40 示。

①RPL:选择器特权,0、1 位;4 种特权

②TI:描述符表指示器;位 2,当 TI=0 时选择全局描述符表 GDT;TI= 1 时选择局部描述表 LDT。

③位 3～15 形成描述符表的入口索引，可寻址为 $2^{13}=8192$ 个描述符。

b. 段描述符的属性位

用段选择器从描述符表中所选择的对象称为描述符。描述符是 8 字节的量，其中包括段的线性基地址和此段的界限(大小)，也包括段的一些属性。这些属性包括此段的保护等级，读、写或执行特权，操作数的默认长度(16 或 32 位)，段的类型以及段的粒度(granularity)，即段的长度的基本单位。描述符中有 12 位包含了有关段的所有属性信息。

图 10-41 表示了一个描述符的一般格式。

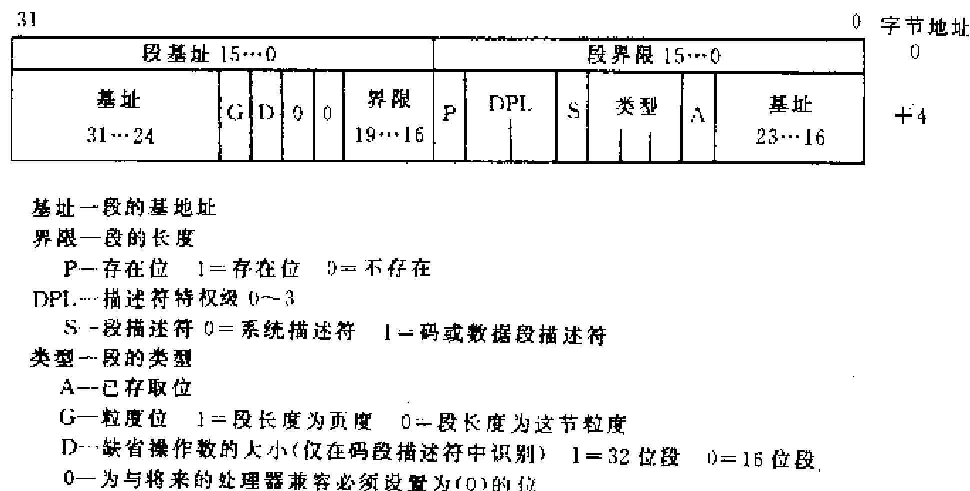


图 10-41 段描述符的一般格式

80386 中所有的段都有 3 个共同的属性场：P 位，DPL 位和 S 位。

P(Present)位即存在位，若此段是物理存储器中，则“存在”位 P=1；若 P=0，此段不在物理存储器中，则任何存取这个段的企图都会引起不存在异常(异常 11)，在异常处理程序中读入此段。

DPL 描述符特权级是一个两位的场，其值为 0～3，确定了此段的保护等级。

S 位反映此段是系统段 S=0，还是用户段即用户程序的码、数据或堆栈段 S=1。系统段与用户段是有很大不同的。

此外在保护方式下，软件可以通过切换进入虚拟的 8086 方式。在该方式下，80386 就像在实地址方式下一样，可执行 8086 的应用程序，而系统程序员仍能利用 80386 的虚拟保护机构。也就是说，在运行 8086 应用程序的同时，可以运行 80386 操作系统及程序。这样，在多用户 80386 中，一个用户可运行 MS-DOS 的电子表格软件，另一用户使用 MS-DOS，而第三个用户可以运行多个 UNIX 实用程序和应用程序。在这种情况下，每一个用户都如同有一个完整的计算机。虚拟 8086 方式为系统程序员提供了很大的灵活性，所以得到很广泛的应用。

第三节 80486 微处理器

一、概述

1990 年 Intel 公司推出了与 80386 完全兼容但功能更强的 32 位微处理器 80486，但它只是对 80386 的底层硬件做了改进，内部操作及寄存器仍是 32 位。该芯片集成了 120 万个晶体

管，以 168 条引线网络阵列式封装，数据线 32 条，地址线 32 条。

80486 时钟频率为 25MHz 和 33MHz。当主频达到 50MHz 时，80486DX 可以在 1 个时钟周期内执行完一条指令。1992 年 80486 DX2 出现，它采用倍频技术，使 CPU 能以双倍于芯片外部的处理速度工作。80486DX2-50，80486DX2-60 是它的代表芯片。这一技术使 80486 的运行速度提高了 70%。

80486 的主要性能：

1. 与大型软件库二进制兼容：

■MS-DOS*，OS/2**，"窗口"；

■UNIX***系统 V/80386；

■iRMX，iRMK™核心；

2. 芯片集成有：

■8k 字节指令和数据超高速缓存；

■浮点部件；

■分页虚拟存储器管理；

3. 使用方便：

■机内自测试；

■硬件测试支持；

■Intel 软件支持；

■扩展的第三者软件支持；

4. 高性能设计：

■常用指令的执行时间为一个时钟周期；

■25MHz 和 33MHz 时钟频率；

■猝发总线传输率 106 兆字节/秒；

■CHMOS/V 半导体技术；

5. 完全的 32 位体系结构：

■地址总线和数据总线；

■寄存器；

6. 多处理器支持：

■具有多处理器指令；

■具有超高速缓存一致性协议；

■支持第二超高速缓存；

80486 与 80386 的主要差别：

为了提高 80486 的性能，在设计时比 80386 增加了如下功能：

1. 减少了执行指令的时钟数，以获得较高的性能。

2. 提高了部线速度。其中包括 1X 时钟、奇偶校验支持、猝发周期、可超高速缓存周期、超高速缓存使无效周期和 8 位总线支持。

3. 为了支持芯片上超高速缓存，控制寄存器增加了一些新位(CD 和 NW)，总线上增加了一些新管脚。在 CR+0 中 CD 和 NW 被复位后，使芯片上超高速缓存被允许。

4. 增加了完整的 80387 数值协处理器的指令集和寄存器组。执行浮点指令期间不执行 I/O 周期。在执行 FINIT/FSAVE 指令后，指令和数据指示字被置"0"，不再出现中断 9，而出现

中断 13。

5. 80486 支持新的浮点出现错误报告方式，以保证与 DOS 兼容。这些新的方式需要在控制寄存器 O 中增加一个新位(NW)和两个新的管脚(FERR#和 IGNNE#)。

6. 增加了 6 条新的指令：

■字节交换(BSWAP)指令；

■交换加法(XADD)指令；

■比较交换(CMPXCHG)指令；

■使数据超高速无效(INVD)指令；

■回写和使数据超高缓存无效(WBINVD)指令；

■使 TLB 条目无效(INVLPG)指令；

7. 控制寄存器 3 中新定义了两位，即页表示目和页面目录条目(PCD 和 PWT)。

8. 增加了新的页面保护特性。这个特性要求在寄存器中新增加一位(WP)。

9. 增加了新的对界检查特性。这个特性要求在标志寄存器中新增加一位(AC) 和在控制寄存器中新增加一位(AM)。

10. 将转换后援缓冲器的替换算法变换成如同芯片上超高速缓存使用的一种为最近最少使用的算法。

11. 增加了三个用于测试片上超高速缓存的新的可测试寄存器 TR3、TR4、TR5 。增强了 TLB 的可测试性。

12. 预取指令排队从 16 字节增加到 32 字节。在修改代码后，始终需要执行一条跳转指令，以保证正确执行新的指令。

13. 复位后，DX 寄存器高阶字节中的 ID 为 04。包括浮点寄存器在内的基地址寄存器的内容在复位后可能有差别。

总之，80486CPU 为 DOS、OS/2、"窗口"和 UNIX 系统 V/80386 的应用提供了最高的性能，它与 80386CPU100%二进制兼容。芯片上 100 万支晶体管集成了超高速缓存、浮点部件和存储器管理部件，与以前的 86 结构系统的其他型号保持二进制兼容。常用指令的执行时间为一个时钟周期，使性能达到了 RISC 水平。在以主频 33.3MHz 工作时，8K 字节的指令和数据兼用的超高频缓存与 106 兆字节/秒的猝发总线结合确保了高的系统总处理能力。

80486 的新的特点增强了多重处理系统，新的指令加速了基于信号灯的存储器控制。片上硬件确保超高速缓存一致性，并为多级超高速缓存提供挂钩。同时，机内自测试广泛地测试片上逻辑电路、超高速缓存和分页转换超高速缓存。调试特性包括执行指令和存取数据时的断点自陷。

二、80486 的基本结构

1. 概念结构

80486 的开发目标是实现高速化并支持处理机。以相当于 386 的 CPU 为核心，除内含高速缓存(Cache memory)和相当于 387 的运算协处理器之外，还采用了易于构成多处理器结构的体制。见图 10-42。

(1)内含高速缓存和浮点处理器

486 内含高速缓存 8KB，可高速存取指令和数据。有关基本指令不用以往的微码控制，而用硬件进行直接控制，在内含高速缓存中通常命中场合下，一条指令所需时钟数平均不超过

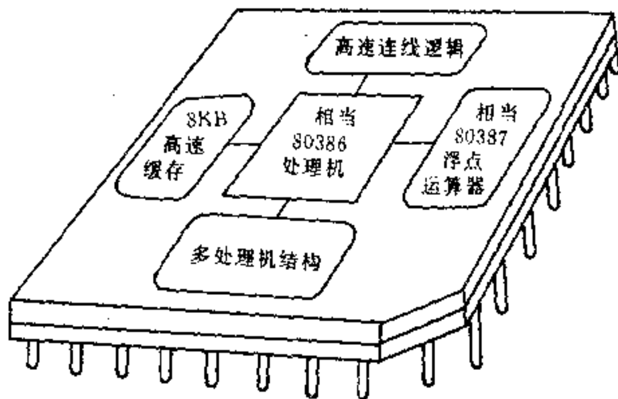


图 10-42　80486 概念结构

1.7 个，比工作在同一频率的 386 速度高 2.5 倍。

浮点运算由于在内含有相当 387 的浮点运算单元(FPU)，比 386 +387 组合的速度高几倍。由于内含 FPU 除减少了 CPU-FPU 间通信之外，而且由于内含高速缓存和 FPU 之间采用 64 位总线连接，其数据传输速度也有很大提高。如果使用 20MHz 的 386，则 486 的速度将比 386 提高 3～4 倍。

(2)沿袭 386 体系结构

从程序人员看，386 体系结构并未改变。至今为止 Intel 公司提供的全部 86 系列微处理机(8086/8088，80186/80188，80286/80386)均保持了目标码级的兼容性，而且与 86/88 的兼容性是以实地址方式保证的。同时，保护地址方式和 386 指标一样。386 引入虚拟 8086 方式在 486 中也继承原样。其考虑的原则是有关工作方式均不改变。但对几种功能进行了扩充，其内部寄存器中的一部分位的内容进行了变动和增加(见后)。指令增加了 6 类。

(3)面向多处理机结构

486 设法得到 RISC 那样的高速处理，在提高单体 CPU 性能的基础上，还可以使用几个 486 构成多处理机的结构，同时设有从其它处理机来的禁止访问共享存储器的访问修改指令。

对芯片本身结构进一步改进工作表现在 Intel 公司与 AT&T 等公司共同合作开发对应处理机的 UNIX 操作系统。该系统将以卡内基梅隆大学(CMU)开发的面向分布式处理的 Mach 操作系统为基础并与 UNIX 系统 V4.0 版相兼容。

2. 80486 内部结构

80486 内部结构如图 10-43 所示。其中整数逻辑单元(ALU)和内存管理单元(MMU)、指令取出/译码单元等都可独立并行工作。386 由 6 个单元构成(总线接口、预取指令、指令译码、执行、控制、MMU)，而 486 还增加了高速缓存和 FPU，用 8 个单元构成。

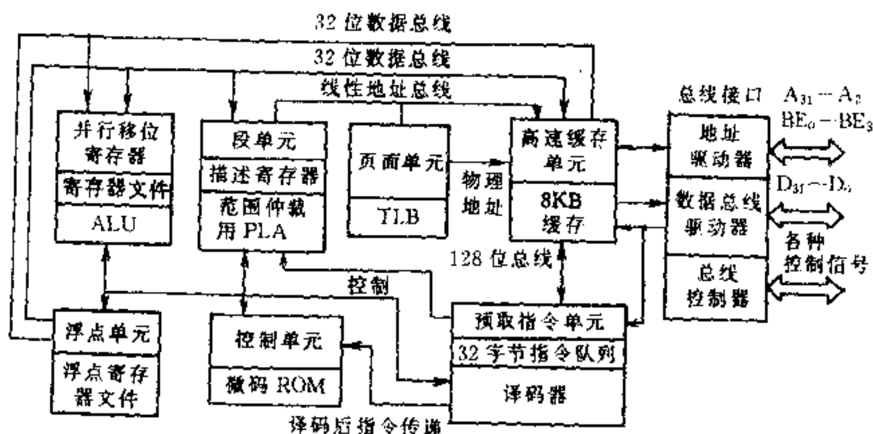


图 10-43　80486 内部结构

(1)基本指令用硬线逻辑执行,内含 128 位总线

486 结构中最重要特点是为提高指令译码的执行速度，在控制单元中采用了一部分硬件控制,同时在高速缓存和预取指令单元之间采用了 128 位总线。

486 在控制单元中仍使用以前的微码结构。其基本指令用硬线逻辑执行。寄存器间的加减法/逻辑运算指令、寄存器和存储器(高速缓存)之间的 MOV 指令等能用 1 个时钟周期执行。而 386 中这些指令的译码执行需要 2～4 个时钟周期。但由于 486 是 CISC 型处理器,其指令数比 RISC 型处理器多许多;全部指令都用硬线逻辑执行是困难的,所以对复杂指令还按以前的方式用微码进行处理。

486 在高速解释执行单元中为迅速传递指令采用了 128 位总线,高速缓存和预取指令单元采用了 128 位总线连接,可以实现一次预取 16 字节的指令,而且预取队列比 386 扩大 2 倍为 32 字节。由于做了这些改进,向译码单元的传送能力为 386 的 8 倍。

(2)高速缓存和 FPU 之间用两条 32 位总线直接相连

486 中内含有相当于 387 的 FPU。在 387 中要用 32 位总线连接存储器;而 486 中的 FPU 浮点寄存器和内含高速缓存是用两条 32 位总线相连的。这两条总线一条作为 64 位总线使用,可以一次把双精度数据(64 位)从高速缓存传送到浮点寄存器,见图 10-43。

386 中采用 387 时还需用其它芯片配合实现，各芯片之间搞得比较麻烦。如从存储器向 387 浮点寄存器装入数据,387 由于没有存储器存取功能,读出数据要靠 386 进行。一旦 386 读出数据,就要改接到 387 之中这些情况是存在的。而 486 可以直接从高速缓存向浮点寄存器传送数据,这样可使时间大幅度减少。然而 486 的浮点指令和 387 完全一样,既没有增加新的也没变动。

(3)四路成组联想高速缓存

图 10-44 为 4 路成组联想式,将 8K 字节的存储器分为 4 组,每个 2K 字节。首先用 4～10 位物理地址选择入口;其次用 11～31 位和各组的地址特征内容比较。凡一致的组即高速缓存命中,0～3 位选择其入口中(16 字节)的特定字节。

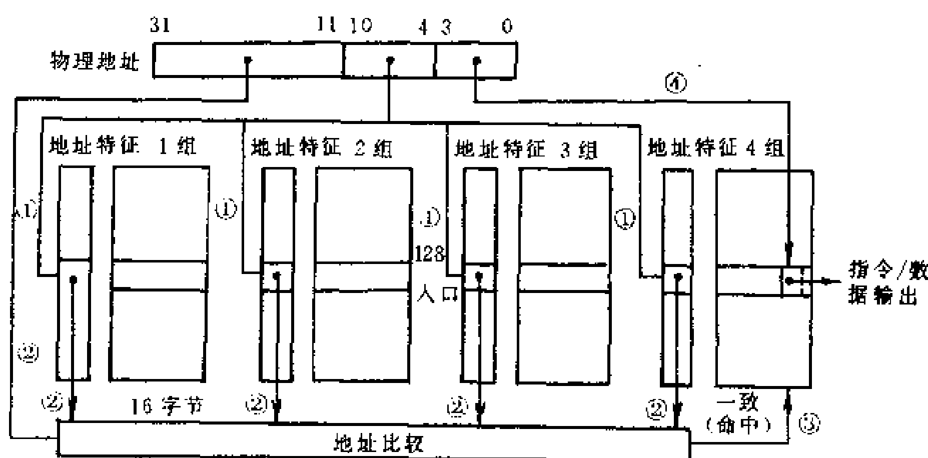


图 10-44　80486 中的高速缓存

因 486 内含 8K 字节高速缓存，指令和数据混合放置，使用的是所谓联合型高速缓存(Unified Cache)。采用 4 路成组联想式高速缓存结构(4 way setassociative cache)，如图 10-44 所示。此时将比直接映像方式的高速缓存的命中率更高。

采用联合型高速缓存的目的是发展多用途，如只执行无数据存取指令时，所有高速缓存就可全部为指令所用；相反，在简单循环中处理大量数据时，在高速缓存中就可以大部分放置数据。

那些无需放在高速缓存中的指令和数据（无命中）可以访问外部存储器。外部存储器中以连接 16 字节为单位装入高速缓存。这时，每一时钟可以迅速传送以 4 字节组成的簇传送方式完成传送。当用 25MHz 时的传送速度为 80Mb/s，是 25MHz 386 能力的 1.6 倍，当高速缓存装满时，会把最近使用的指令和数据优先留在高速缓存之中。采用所谓 LRU 方法(least recently used)，找出今后使用可能性差的那些指令和数据。

80486 技术指标如表 10-9 所示。

表 10-9　80486 技术指标

项　　目	指　　标
半导体工艺技术	1.0μmCHMOS
集成晶体管数目	约 120 万晶体管
芯片尺寸	约 16×11mm
外部数据总线	32 位
外部地址总线	32 位
时钟频率	25MHz/33.3MHz
性能(25MHz)Dhrystone 值	37000Dhrystones/s
双精度 Wbetstone 值	6.1MWIPS
高速缓存容量	8KB(指令/数据兼用)
流水线级数	5 级
寄存器	和 80386，80387 兼容
逻辑地址空间	64TB
物理地址空间	4GB
引线数	168 条，PGA 型
功耗	最大 4.5W(24MHz 时)

(4) 标志寄存器和控制寄存器扩充

另外在保护地址方式中，486 中如未使用新定义的标志和中断向量，则 286/386 所用的程序可照样执行。协处理器 287/ 387 所用的程序也一样可用。协处理器的有无、对协处理器的种类的掌握，可用控制寄存器等确认协处理器识别程序进行。当不能连接协处理器时，仿真浮点运算的程序也可以在 486 上工作，但对 486 的标志位和控制寄存器中内容要进行修改，见图 10-45～图 10-47。

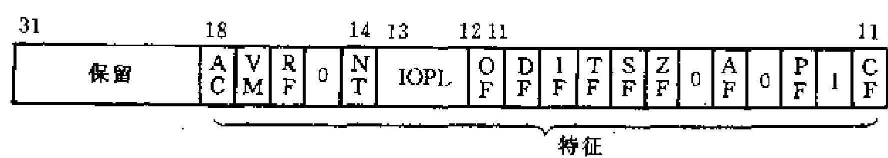


图 10-45　扩充标志寄存器(EFLAGS)改动之点

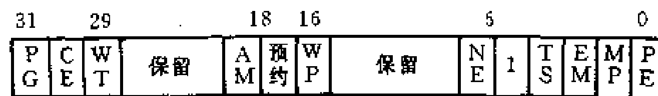


图 10-46　控制寄存器 CR 的变动内容

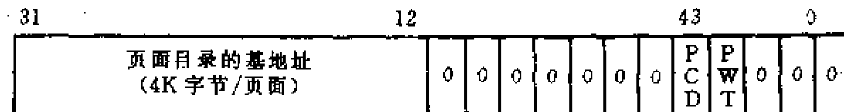


图 10-47　控制存储器 CR3 的变动内容

■AC 位(alingment check 定位检查):AC＝"1"时,不依据数据属性的定位(字边界和全局字边界)访问存储器时,产生定位故障(alignment fault)但该故障只发生特权级 3(用户方式),不会发生在比其高的特权级中。AC＝"0"时,不进行定位检查,与 80386 工作兼容。定位故障例外 17 是被定义的新中断。

■CE 位(cache enable 高速缓存工作):决定内含高速缓存是否有效。CE＝"0",高速缓存无效,即使错误命中发生,也不取指令/数据(高速缓存填入),CE＝"1"时,高速缓存有效,误命中时,从外部存储器取指令/数据。但 CE 位同 KEN(高速缓存工作)信号和 CR3 寄存器中的 PCD 位有关,误命中时,高速缓存如何填写内容,由这些状态来决定。

■WT 位(write transparennt 写透明):数据写入时,决定高速缓存命中发生时的动作。WT＝"1"时,对高速缓存和外部存储器写入数据(写到底)。WT＝"0"时,保护对外部存储器写入数据。

■AM 位(alignment mask 定位屏蔽),EFLAGS 中的 AC 位为"1"时,决定是否产生定位故障。AM＝"0"时,与 80386 兼容。以 80386 为前提作的应用程序操作 EFLAGS 中未定义位(如 AC 位)可以不用考虑。80486 中当 AM＝"1",AC＝"1"时,如不根据定位即对存储器存取,就将发生故障。

■WP 位(write protect 写保护):在 80386 中即使只允许页面读出,则在监控(supervisor)方式下,也允许写入。在 80486 中,为了禁止这一点而定义了 WP 位。WP＝"1"时,对专用页面读出的同时也作写入时,则将发生故障。

■NE 位(numerics-exception 数值例外):浮点运算单元中,没被屏蔽发生例外时,选择使用例外 16,或是使用其它外部中断。NE＝"1"时,浮点运算单元的例外,以例外向量 16 通知;NE＝"0"时,同 MS-DOS 系统中以前所用的协处理器的控制之间保持有兼容性。

■MP 位(monitor processor 监控处理器):在 80286/80386 中有协处理器时,MP＝"1";不存在时 MP＝"0"是用软件设置。在 80486 中由于没有 Busy 信号,MP 位是无意义的。但为了保持软件兼容性而保留该位。在 80486 归零时 MP＝"0"。

■PWT 位(page write-thiought 写页面到底):在当前访问的页面中 PWT＝"1"时,写到底;WPT＝"0"时,写返回。在 80486 中内含高速缓存,写到底为专用,PWT 内容被忽略。当外加高速缓存(二次高速缓存)时,采用写返回方式,依据该位内容可以控制向主存写入的定时。

■PCD 位(page cache enable 页面高速缓存工作):内含高速缓存决定以页面为单位是否有效。只有 KEN(高速缓存工作)信号工作,PCD＝"0"时,内含高速缓存才有效工作。PCD＝

"1"时，只对外部高速缓存(或外部存储器)进行读出/写入。

三、80486 的引脚

80486 外露引脚有 168 条，按功能分有地址总线、数据总线和控制总线。详细分类如图 10-48 所示。从图 10-48 看出，32 位地址总线是用 30 位地址线 A_2～A_{31} 加上 4 个字节允许符来实现的，这 4 个字节允许符给出了 2 个最低有效地址位和传送宽度编码。有许多控制信号，像 $M/\overline{IO}$(存储器/输入输出)、$D/\overline{C}$(数据/控制) $W/\overline{R}$(读/写)、$\overline{RDY}$(准备就绪)、$\overline{LOCK}$、HOLD 以及 HLDA(保持确认)等，都与 80386 的对应信号相似，而像 RESET、NMI 和 INTR 输入则完全与 80386 相同。80486 像 80386 一样通过BS16(总线大小 16 位)信号为 16 位设备提供动态总线大小，并为支持使用一字节宽度的设备增加了一个BS8(总线大小 8 位)信号。

80486 微处理机也有类似 80386 微处理机的总线保持特性。在总线保持期间，80486 微处理机通过浮空地址、数据和控制总线而放弃本地总线的控制。除总线保持外，80486 微处理机还有地址保持特性。在地址保持期间，只有地址总线被浮空，数据和控制总线仍保持激活。地址保持用于高速缓存的线路无效性控制。

现对 80486 的管脚功能作如下说明：

(1)时钟(CLK)

CLK 为 80486 微处理机提供基本的定时和内部的工作频率。所有的外部定时计数都是相对于 CLK 的上升沿指定的。80486 微处理机可以在很宽的频率范围内工作，当 RESET 无效时，CLK 的频率不能迅速改变。为保证芯片工作正常，CLK 的工作频率必须稳定。CLK 只需要 TTL 电平来正常工作。

(2)地址总线(A_2～A_{31}，BE_0～BE_3)

A_2～A_{31} 和 BE_0～BE_3 形成地址总线，并提供内存和 I/O 端口的物理地址。80486 微处理机能寻址 4GB 的物理内存空间，以及 64KB 的 I/O 地址空间。A_2～A_{31} 用来寻址到一个 4 字节的单元。BE_0～BE_3 则用来标识在当前的传送操作中要涉及 4 字节单元中的哪些字符。对于外部的内存储器执行读和写周期时，字节使能输出 BE_0～BE_3 用来确定哪些字节必须被驱动有效。BE_3 适用于 D_{24}～D_{31}，BE_2 适用于 D_{16}～D_{23}，BE_1 适用于 D_8～D_{15}，BE_0 适用于 D_0～D_7。

(3)数据总线(D_0～D_{31})

D_0～D_{31} 为 80486 微处理机的双向数据总线。D_0～D_7 定义为最低字节，D_{24}～D_{31} 定义为最高字节。可以使用由 BS8 和 BS16 输入引脚控制的确定数据总线宽度特性，将数据传送到 8 位或 16 位的设备中。

(4)奇偶校验(DP_0～DP_3，PCHK)

奇偶校验共两组信号，一组是 DP_0～DP_3，另一组是 PCHK。

DP_0～DP_3 这四个是奇偶数据通路(输入/输出)管脚。偶数奇偶校验意味着：在 8 个相应的数据总线引脚和奇偶校验引脚上，有偶数个输入为高电平。

PCHK 为奇偶校验状态输出。这个引脚为低电平时表示一个偶错。

(5)总线周期定义

M/IO，D/C，W/R 输出

M/IO，D/C 和 W/R 是一些主要的总线周期定义信号。M/IO 用来区别内存和 I/O 周期；D/C 用来区别数据和控制周期；W/R 用来区别写周期和读周期。

LOCK 总线锁定输出

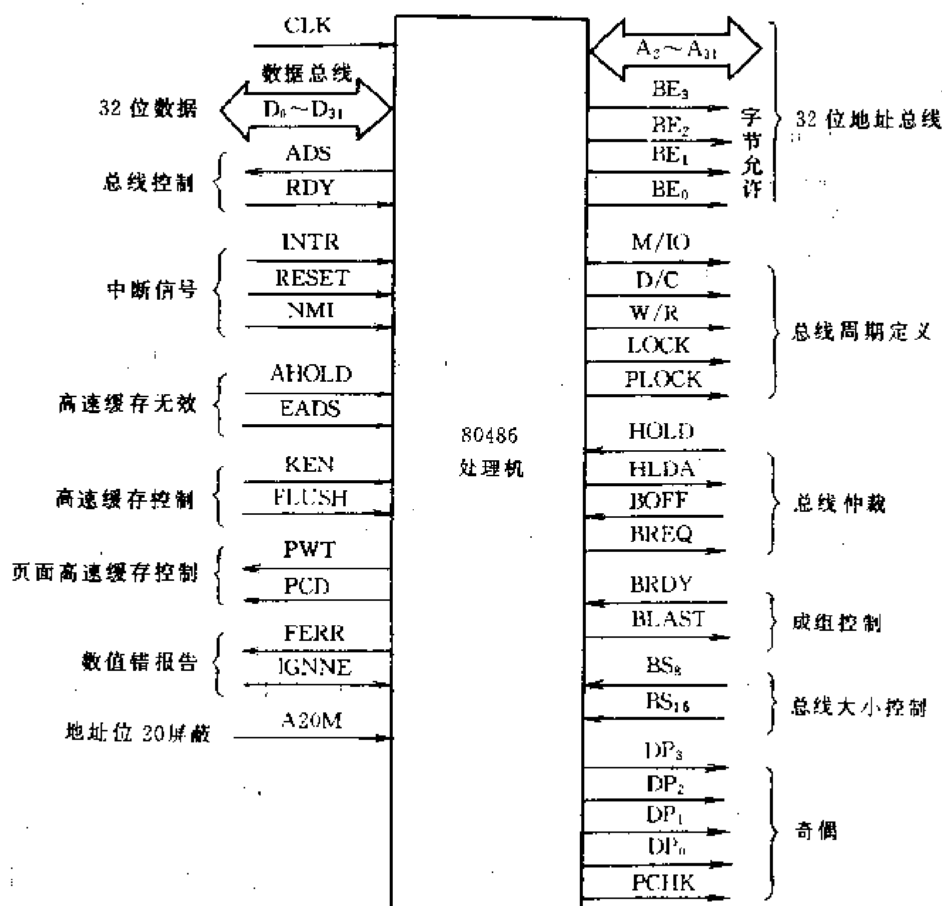


图 10-48 80486 的引脚功能信号图

LOCK 表明 80486 微处理机正在读一改一写周期中运行，在写周期和读周期之间，不得放弃外部总线。当发送 LOCK 时，当前的总线周期被锁定，允许 80486 微处理机独占系统总线的访问。当发出 LOCK 时，80486 微处理机将不确认总线保持。LOCK 是低电平有效，且在总线保持期间被浮空。

PLOCK 伪锁定输出

这是 Intel 定义的伪锁定输出信号，同时 Intel 指定把这个信号当成总线周期的定义。与 LOCK 位不同的伪锁定表示是临界的读-改-写操作。在这种操作中，在字节操作完成前 有其它的系统部件检查被修改了的项。

(6)总线控制

总线控制信号允许处理机去指明总线周期是何时开始的，并允许其它系统硬件去控制数据总线宽度与总线周期的终止。

ADS 地址状态输出

ADS 输出指明地址和数据周期定义信号均有效。这一信号在总线周期的第一时钟周期内激活，在该周期的第二个及后续的时钟周期内变为无效。ADS 低电平有效，且在总线保持期间不驱动，它可提供外部总线电路用作一指示，表明处理机已启动总线周期。

RDY 输入

RDY 表明当前的总线周期是完整的。响应读请求时，RDY 表明外部系统已在数据引脚放好了有效数据。而在响应写请求时，RDY 则表明外部系统已接收了 80486 微处理机的数据。

RDY 低电平有效，内部没有上拉电阻。这一输入必须满足建立和保持时间的要求，以保证系统正常工作。

(7)成组控制

BRDY 成组准备就绪

这个输入引脚表示当前周期已经完成。

BLAST 最近成组输出

它表示现在进行的是成组传送方式

(8)高速缓存控制

由于把高速缓存以及高速缓存控制器都集中到 80486 片内，所以高速缓存控制信号很丰富。

KEN 高速缓存允许引脚

它用来确定当前周期所传送的数据是否可高速缓存。当 KEN 信号有效(输入为低电平有效)，并且 80486 微处理机产生一个可高速缓存的周期时，该周期将被控制成高速缓存行组填入周期。

FLUSH 输入

强制 80486 微处理机清洗其整个内部高速缓存。FLUSH 低电平有效，且只需持续一个时钟的时间。

(9)高速缓存的无效性控制

在高速缓存的无效性控制周期里，使用 AHOLD 和 EADS 输入。AHOLD 是 80486 微处理机地址线 A_4～A_{31}能否接受地址输入的一个制约条件。EADS 表明地址输入端上的外部地址是实际有效的。激活 EADS 后，将使 80486 微处理机去读外部地址总线，并对指明的地址执行内部的高速缓存的无效性控制周期。

(10)页面高速缓存控制

PWT 页写贯穿。

PCD 页高速缓存禁止。

这两个信号反应了内部寄存器的置位情况。当 KEN 允许硬件控制存储器的专用物理区域进行高速缓存时，这两个引脚表明在逻辑存储器各页上已经使用软件对高速缓存进行了控制。

(11)数据出错报告

FERR 和 IGNNE 报告浮点错。

FERR 浮点错。

这个输出信号类似于 80387 的浮点错信号，而且是在确定的条件下使用，产生中断 13。

IGNNE 不理采数据处理器错

若没有软件恰当地激活，IGNNE 输入引脚是无效的。

(12)地址位 20 的屏蔽

发出 A20M 后，将时 80486 微处理机在内部高速缓存中执行查找之前，以及驱动一个内存周期到外部去之前，屏蔽第 20 位物理地址。

(13)总线仲裁

BREQ 总线请求输出

每当总线周期在内部执行中时，80486 就发出 BREQ。

HOLD 总线保持请求输入

它允许另一个总线主设备完成 80486 微处理机总线控制。

HLDA 总线保持确认输出

它表明 80486 微处理机已经将总线交给另外一个本地的总线设备。HLDA 变为有效，以响应 HOLD 引脚上出现的保持请求。当脱离总线保持时，HLDA 被驱动至无效状态，而 80486 微处理机将恢复其对总线的驱动。

BOFF 输入

强制 80486 微处理机在下一个时钟周期间释放其对总线的控制。

(14)总线大小控制

BS8 和 BS16 控制总线宽度，它借助于少量的外部元件，就能支持外部的 16 位和 8 位总线。80486 的 CPU 每个时钟都采样这些引脚。当发送 BS16 和 BS8 时，只需要 16 位和 8 位的数据总线有效。如果同时发 BS16 和 BS8，则选用 8 位的总线宽度。

(15)中断信号

80486 微处理机在中断上有一个 2 时钟的同步电路。在发出 INTR 之后 2 个时钟内，中断请求将到达内部的指令执行单元。INTR 信号是电平型的，为使指令执行单元能识别它，它必须保持有效。如果 INTR 信号不保持有效，那末 80486 微处理机将不执行中断服务。

NMI 为边沿触发，NMI 信号的上升沿被用来产生中断请求。NMI 输入不需要在中断被实际服务之前一直保持有效。

RESET 为复位信号，当发出 RESET 之后，80486 微处理机各个寄存器的值如表 10-10 所示。表中 BIST 为复位期间 80486 微处理机运行的内存的自测试功能。在 RESET 之后，80486 微处理机将从 FFFFFFF0H 单元开始执行指令。

表 10-10 复位后各寄存器的值

计存器	初始值(BIST)	初始值(NO BIST)
EAX	0(通过)	不定
ECX	不定	不定
EDX	0400+版本 ID	0400+版本 ID
EBX	不定	不定
ESP	不定	不定
EBP	不定	不定
ESI	不定	不定
EDI	不定	不定
EFLAGS	00000002H	00000002H
EIP	0FFF0H	0FFF0H
ES	0000H	0000H
CS	F000H	F000H
SS	0000H	0000H
DS	0000H	0000H
FS	0000H	0000H
GS	0000H	0000H

（续）

计存器	初始值(BIST)	初始值(NO BIST)
IDTR	基值=0,界限=3FFH	基值=0,界限=3FFH
CRO	60000000H	60000000H
DR7	00000000H	00000000H
CW	037FH	不变
SW	0000H	不变
TW	FFFFH	不变
FIP	00000000H	不变
FEA	00000000H	不变
FCS	0000H	不变
FDS	0000H	不变
FOP	000H	不变
FSTACK	不定	不变

四、80486的多处理机基本结构

Intel公司设想的多处理机结构如图10-49所示，它由486外部高速缓存和总线控制电路等组成的CPU板接在高速系统总线上构成。这种系统总线传输率最低为80Mb～100Mb/s。目前的微通道和EISA都还达不到这种能力，CPU板都可共享主存储器。共享主存采用并行处理同步通信方式。为减轻系统总线的负载，CPU板上设有大容量的外部高速缓存。由于CPU本身自含高速缓存，这种外部高速缓存称为二次高速缓存。

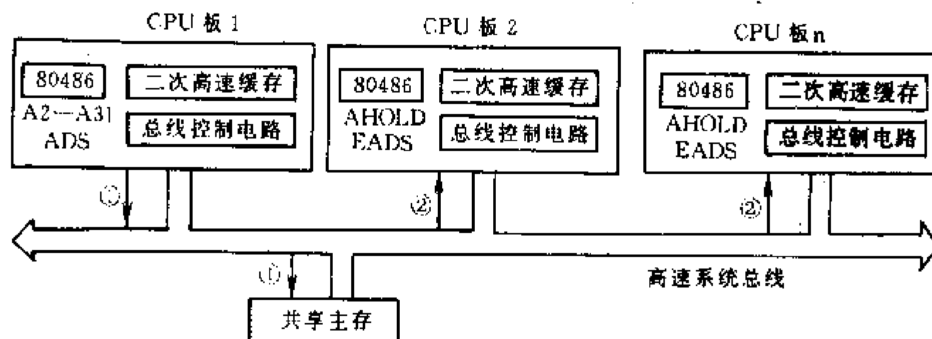


图10-49　80486的多处理机结构

可能会发生这种情况，即当某CPU板与共享主存交换了信息后，其它CPU板复制在本地高速缓存中的某CPU信息也需要更新。因此在486中设有知道要和共享主存交换信息的AHOLD信号，以及为了接受被交换的共享主存地址所设置的定时信号EADS信号两条引线(见图10-49)。所谓总线抽样机构，在486中就是：如果输入AHOLD信号，486上输出到地址总线上的地址则用EADS信号进行锁存。而且该地址信号如已存在于内含高速缓存之中，则令其无效。为利用这种工作机制，指定共享主存的地址在总线上流动时，必须把AHOLD信号和EADS信号输入给486的地址译码电路。

为了访问共享主存等，新设有请求系统总线的 BREQ 信号。对该信号如无响应信号，则用 BOFF 表示。BOFF 信号与以前的 HOLD 信号类似，当执行中指令结束时，HOLD 信号如没能停止 CPU 工作，则 BOFF 信号以时钟为单位可以停止 CPU 工作。486 中用每个时钟监视 BOFF，如 BOFF 信号被输入就位（用下个时钟），则地址、数据、状态信号都将处于浮动状态，而总线的输出立即完全终止。如果其它处理机使用总线结束，被中断的总线周期将返回初始状态再启动。这样，总线使用中断的单位要细分，需要提高高速总线使用的响应速度。

五、80486 的工作方式

80486 有如图 10-50 所示的三种工作方式，即实地址方式（REAL），保护方式（PROTECTED）和虚拟方式（VIRTUAL8086）。如果对 CPU 进行复位，就进入实地址方式工作。修改 CR0 和 MSM 控制寄存器，80486 就由实地址方式转移到保护方式，或进行其反方向的方式转移。执行 IRETD 指令或进行任务转换，就由保护方式转移到虚拟的 8086 方式。任务转换功能是 80486 的特点之一。采用中断 CPU 就有可能由虚拟的 8086 方式返回到保护方式。

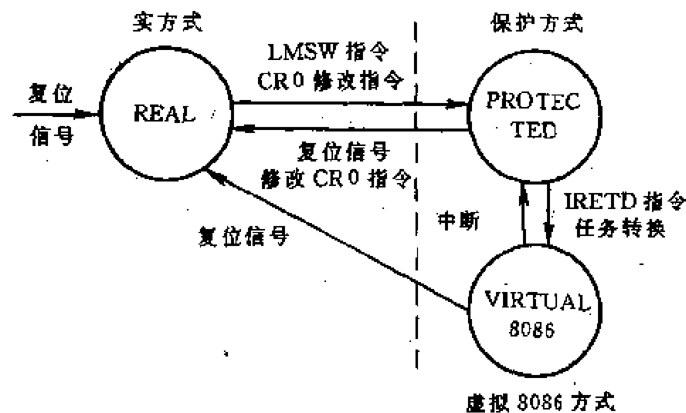


图 10-50　80486 的三种工作方式

80486 的实地址方式的工作原理与 8086 基本相同，主要区别是 80486 能处理 32 位数据。在保护方式下，CPU 可访问 2^{32}字节的物理存储空间，段长为 2^{32}字节。而且也可以实施保护功能。分页功能是任选的。在这种方式中，引入了软件可占用空间的虚拟存储器的概念。

虚拟的 8086 方式是一种既能有效利用保护功能，又能执行 8086 代码的工作方式。CPU 与保护方式下工作原理相同，但程序指定的逻辑地址与 8086 相同解释。

第四节　Pentium 微处理器

一、概述

1993 年 3 月 INTEL 公司推出了当前最先进的微处理器芯片——64 位的 Pentium，这是第五代微处理器。该芯片集成了 310 万个晶体管，有 64 条数据线，36 条地址线。

Pentium（以下称 P5）采用了新的体系结构，与 80486 相比有以下重要改进：

■超级标量体系结构

■动态转移预测

■流水线浮点部件

■较大容量的片上超高速缓存(增加 8K)

■较强的错误检测和报告功能

■更多的测试挂钩(边界扫描、碳针方式)

P5 以与 INTEL 486 CPU 相同的频率工作时,整数运算性能提高一倍,浮点性能提高 5 倍。P5 的初始目标频率是 66MHz。

P5 的两条流水线和浮点部件能够独立工作。每条流水线在一个时钟内发送一条常用的指令。这两条流水线在一个时钟内可以发送两条整数指令,或在一个时钟内发送一条浮点指令(在某些情况下可以发送两条浮点指令)。

浮点部件在 INTEL 486 基础上完全重新进行了设计。快速算法可使诸如 ADD,MOD 和 LOAD 等公用运算的速度最少提高 3 倍。许多应用程序中利用指令调度和重叠(流水线)执行可以使性能提高 5 倍或更多。

P5 设计成能利用所有 C5C/C8C 超高速缓存的功能。

P5 实现了一种 36 位地址总线,同时该结构也支持 64 位的物理地址空间。

二、Pentium 的功能结构

图 10-51 所示为 P5 的内部结构框图。由图看出:P5 有两条指令流水线,即“U ”流水线和“V”流水线。U 和 V 流水线都执行整数指令。但只有“U”流水线执行浮点指令。在“V ”流水线中也可以执行一条异常的 FXCH 指令。因此,P5 能够在每个时钟内执行两条整数指令,或在每个时钟内执行一条浮点指令。如果两条浮点指令中有一条为 FXCH 指令,那么在一个时钟内

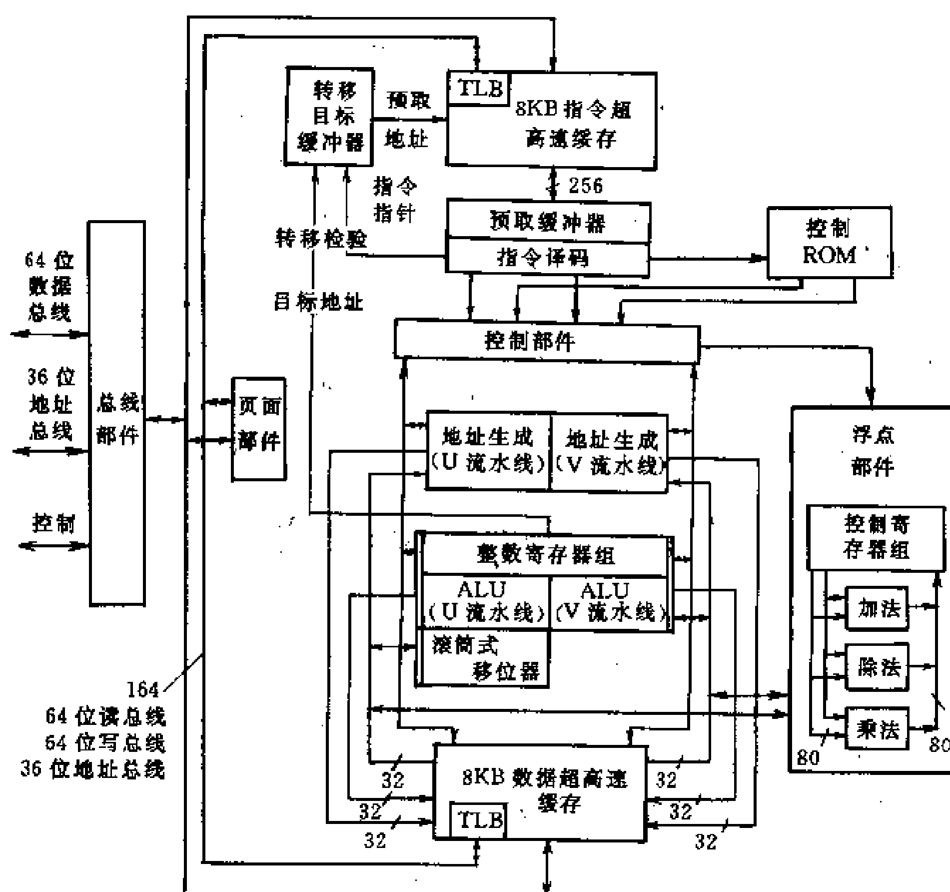


图 10-51　P5 内部结构框图

可以执行两条浮点指令。每条流水线都有自己独立的地址生成逻辑、算术逻辑部件和数据超高速缓存接口。

如图 10-51 所示，P5 有两个独立的超高速缓存，即一个指令超高速缓存和一个数据超高速缓存。数据超高速缓存有两个端口，分别用于两条流水线。它有一个专用的转换后援缓冲器(TLB)，用来把线性地址转换成数据超高速缓存所用的物理地址。指令超高速缓存、转移目标缓存器和预取缓存器负责将原始指令送入 P5 的执行部件。指令取自超高速缓存或外部总线。转移地址由转移目标缓冲器予以记录，指令超高速缓存的 TLB 将线性地址转换成指令超高速缓存器所用的物理地址。译码部件将预取的指令译码成 P5 可以执行的指令。控制 ROM 含有控制实现 P5 体系结构必须执行的运算顺序的微代码。控制 ROM 部件直接控制两条流水线。

P5 对 80486 的寄存器作了如下扩充：

标志寄存器增加了两位：VIF、VIP。此两位用于控制 Pentium 的虚拟 8086 方式扩充部分的虚拟中断。控制寄存器中增加了一个 CR4，，并对 CR0 中的 CD 位和 NIV 位的含义进行了重新定义。增加了几个模型专用寄存器，用来控制可测试性、执行跟踪、性能监测和机器检查错误功能等。Pentium 可用指令读写这些寄存器。

三、Pentium 的引脚

P5 共有 168 个引脚，可分处理器引脚，总线接口引脚，调试和测试引脚。下面分别加以说明。

1. 处理器控制引脚

P5 处理器的控制引脚见表 10-11。

表 10-11 处理器控制引脚

类 型	引 脚 名	数 量
时钟	CLK	1
初始化	INIT，RESET	2
FRC	FRCMC#，IERR#	2
总线仲裁	BOFF#，BREQ，HOLD，HLDA	4
超高速缓存窥视	AHILD，EADS#，FLUSH#，HITM#，HIT#，INV	6
中断	INTR，NMI	2
执行跟踪	IU，IV，IBT	3
数字错误	FERR#，IGNNE#	2
系统管理	SMI#，SMIACT#	2
其他	A20M#	1

CLK 输入决定 P5 的执行速度和定时。引脚定时根据这个信号的上升边规定 CLK 信号的目标速度是 66MHz。P5 的正常操作要求 CLK 为 TTL 电平。

RESET 输入使 P5 从已知状态开始执行。P5 从 0FFFFFFF0H 地址开始执行。当 RESET 建立时，内部超高速缓存、转换后援缓冲器(TLB)、转移目标缓冲器(BTB)和段描述符超高速缓存(SDC)无效。

INIT 输入类似于 RESET 引脚，其中一个例外是有效的 INIT 不干扰内部超高速缓存、模型专用寄存器或浮点状态(使 TLB 和 BTB 无效)。同样，一旦建立则立即识别 RESET，而 INIT 为一次中断并因此根据指令边界予以识别。INIT 可用来帮助执行为 80286 编写的 DOS 扩展程序。INIT 提供了一种从保护方式转换到实方式而又保存内部超高速缓存内容的方法：当 RESET 从高电平转换到低电平以决定是否应该执行内部自测试(BIST)时对 INIT 取样。

FRCM#和 IERR#引脚用以支持功能冗余检查(FRC)。FRCM#引脚决定 P5 是否以主方式或检验器方式配置。IERR#引脚有效则表示主 CPU 和检验器 CPU 之间不匹配或表示内部出错。

在每一个 RESET 有效的时钟内取样 FRCM#引脚。若复位期间取样为低电平，P5 的输出为三态(除 IERR#之外)。当 RESET 从高电平转换到低电平以决定 P5 是否配置为主方式或检验器方式时，则对 FRCM#取样。若 FRCM#的取样为高电平，P5 配置成主方式。P5 在主方式时按照总线协议的要求驱动其输出引脚。若 FRCM#取样为低电平，P5 配置为检验器。P5 在检验器方式时其所有的输出都为三态输出(除 IERR#外)，然后对通常用主方式驱动的输出引脚取样。若取样值与内部计算值不同，检验器 P5 建立 IERR#以表示出错误。在检测不匹配后的两个时钟内建立 IERR#，并且在每次不匹配而被建立时占用一个时钟。注意：在 RESET 后配置为主方式/检验器方式，并且除接着的 RESET 之外都不可能加以改变。

IERR#引脚有第二种功能。它也可以建立以表示检测出 CPU 内部阵列的读奇偶错。

HOLD 信号使 P5 释放对总线的控制。若 HOLD 建立，P5 将在完成所有未完成总线周期后释放总线。当 P5 释放其对总线的控制时，它驱动 HLDA 信号使其活动。当保持有效时，只要内部超高速缓存满足存储器请求，P5 不可以继续执行。

P5 提供一个 BOFF#输入，用以防止多主机系统中的死锁。若这个信号被建立，P5 将在下一个时钟释放其对总线的控制(不要求确认)。那些在总线 HLDA 期间被浮动的同样信号在 BOFF#期间也被浮动。在 BOFF#建立时任何连续的总线周期都被中止。只要 BOFF#保持活动，总线就会保持高阻抗状态。BOFF#在建立前返回的数据由处理器用于当前周期，但有写入超高速缓存。当使 BOFF#无效时，P5 重新启动全部被中止的总线周期。这与 INTEL 486 CPU 的实现方法有所不同。

无论何时 P5 请求一个总线周期时，它就会驱动 BREQ 引脚为活动的。这可用于智能总线仲裁。不论是在总线保持状态还是地址保持状态，这个引脚始终被驱动。AHOLD 信号用来调节 P5 的 A_{35}～A_3 和 AP 地址线以输入地址。

AHOLD 一般用以启动询问周期(询问周期与窥视周期相同)。P5 将在紧跟 AHOLD 被建立的时钟后面的时钟内浮动地址总线，因此不需要确认。因为窥视影响整个超高速缓存行，所以虽然在 AHOLD 期间 A_{35}～A_3 被浮动，但 P5 只将 A_{35}～A_5 用于询问周期和奇偶校验。地址引脚 3 和 4(A_3,A_4)在询问周期期间从逻辑上予以忽略并且不作为地址奇偶校验部分进行检验，但必须在有效的逻辑电平上取样。当地址总线浮动时，P5 的其余总线将保持活动，这样来自早已忙碌的总线周期的数据事务处理可以继续下去。

伴随着输入 EADS#表明外部窥视地址在 P5 的地址输入上是有效的。要运行询问周期，外部器件必须确保 P5 地址总被浮动(AHOLD,HLDA 或 BOFF#为活动的)，驱动窥视到 A_{35}～A_5 上的地址并建立 EADS#。外部器件还驱动 INV，这表明窥视是否应该使该超高速缓存单元无效(INV 建立)，或仅仅表示其为共享(INV 撤销建立)。根据该询问周期，若该特殊存储单元目前保存在 P5 的超高速缓存里，那么 P5 建立 HIT#。另外，若该地址与修改匹配，P5

建立 HITM # 并调度回写周期把修改行写出到总线。

FLUSH # 引脚用来使 P5 回写数据超高速缓存中所有修改的行以及内部指令和数据超高速缓存的内容无效。当回写和使无效结束时，运行刷新确认的特定周期。

无论何时遇到未屏蔽的浮动错误，P5 都建立 FERR #。FERR # 类似于 INTEL387 数值协处理器上的 ERROR # 引脚。FERR # 可由外部逻辑用于 P5 系统中报告的 PC 型浮动错误。FERR # 低电平有效，并且在总线保持期间不被浮动。

在建立 IGNNE # 时，P5 将忽略数值错误并继续执行非控制浮点指令。在撤销建立时，若先前的指令造成错误，P5 将冻结非控制浮点指令。当控制寄存器 0 中的 NE 位置位时，IGNNE # 不起作用。IGNNE # 输入低电平有效。P5 内部没有连接上拉电阻。

P5 实现了一种可跟踪指令执行的机构，并通过三个输出引脚 IU，IV 和 IBT 与总线协同动作。IU 输出有效时表明 U 流水线中的指令已结束执行。同样，IV 输出有效表明 V 流水线中的指令已完成执行。另外，进行转移时 P5 提供活动的 IBT 输出。若测试寄存器 12 中执行跟踪的允许位置位，转移跟踪信息特定周期将根据每个 IBT 的建立运行。

P5 实现了一种对 INTEL 386 体系结构的扩展，称作系统管理方式(SMM)。进入这个方式时，P5 的状态被保留起来，并允许一个单独的存储空间。SMM 可用来实现权力管理功能，并以一种对软件透明的方式提供其他一些特性(例如安全)。有两个引脚用来支持 SMM，SMI #(系统权力管理中断)和 SMIACT #(系统权力管理中断活动)。SMI # 是用以请求 SMM 的下降边敏感中断。SMIAT # 输出活动，表明 P5 正以系统管理方式工作。

建立 A20M # 输入使 P5 屏蔽用来在内部超高速缓存中执行查找地址的地址位 20，并将存储周期排出到外部世界。当地址根据询问周期被驱动入 P5 时，不管 A20M # 的状态如何，地址位 20 都不会被屏蔽。当 A20M # 被建立时，P5 仿真 8086 中的 1M 字节地址的环。A20M # 为低电平活动并且只有当处理器处在实方式时才可能被建立。保护方式期间对 A20M # 的建立未予定义。

2. 总线接口引脚

P5 的总线接口引脚见表 10-12。

表 10-12 总线接口引脚

类 型	引 脚 名	数 量
地址总线	A_{35}～A_3	33
地址奇偶校验	AP	1
地址奇偶校验错误	APCHK #	1
字节允许	BE_7 # ～BE_0 #	8
数据总线	D_{63}～D_0	64
数据奇偶校验	DP_7～DP_0	8
数据奇偶校验允许	PEN #	1
数据奇偶校验错误	PCHK #	1
总线检查	BUSCHK #	1
页面属性	PWT、PCD	2
总线周期定义	D/C #，W/R #，M/IO #，SCYC	4
总线周期长度	CACHE #	1

（续）

类　型	引　脚　名	数　量
总线封锁	LOCK#	1
总线周期控制	ADS#、BRDY#	2
超高速缓存允许	KEN#	1
下一地址	NA#	1
回写/通写	WB/WT#	1
外部与缓冲器空	EWBE#	1

总线接口有 64 个传送数据的双向数据引脚(D_{63}～D_0)。33 位地址总线(A_{35}～A_3)标识一个 8 字节单元的地址。在 8 字节单元内每个字节的单独字节允许引脚(BE7#～BE0#)标识用于特定访问的该字节或若干字节。对写来说，字节允许引脚应始终用来决定应写入哪些字节。对读来说，这些引脚表示请求的是哪个或哪些字节。对于那些可以超高速缓存的读来说，所有 8 个字节必须有效返回而不管 BE7#－BE0#引脚的状态。

地址总线(A_{35}～A_5)有一个奇偶校验位。利用与 P5 驱动的地址相同的定时在所有 P5 总线周期上产生偶数地址奇偶校验(AP)。AP 引脚在建立 EADS#的任一时钟内取样。若 P5 在询问周期期间在地址总线上检测出奇偶错误，则在 EADS#取样有效后两个时钟建立 APCHK #。询问周期不受地址奇偶错误的影响并将继续到结束。

数据总线的每个字节都有一个奇偶校验位。利用与 P5 驱动的数据相同的定时在所有写数据周期上产生偶数数据奇偶校验(DP_7～DP_0)。读周期时校验偶数数据奇偶校验。PCHK#引脚在 BRDY#返回处理后两个时钟反映奇偶校验的结果。PCHK #为低电平时表示奇偶错误。PEN#输入决定机器检查中断是否将作为读周期期间数据奇偶错误的结果。

BUSCHK#输入为系统提供一种用信号表示总线周期的失败结束。这个信号在为读和写建立 BRDY#的任意边沿上取样。若这个输入取样建立，P5 将锁存周期地址和控制信号。另外，若 CR4 中的机器检查异常位置“1”，P5 将在完成指令时导向机器检查异常。BUSCHK#用来指出与总线周期同步的错误。

PWT(页面通写)和 PCD(页面超高速缓存禁止)是 CR3(控制寄存器 3)、PDPE(页面目录指示字项)、PDE(页面目录项)和 PTE(页表项)中定义的两位。PWT 控制通写策略。PWT＝1 时定义当前页面的通写策略。PWT＝0 允许回写的可能性。PCD 控制可超高速缓存性。PCD＝0 允许在片上超高速缓存。PCD 本身不允许超高速缓存，它必须由控制寄存器 0(CR0)中的 KEN#(超高速缓存允许)输入信号和 CD(超高速缓存禁止)位状态控制。PCD＝1 时，不管 KEN#和 CD 的状态如何都超高速缓存。PCD 位和 PWT 位的状态都在存储器访问期间在 PCD 和 PWT 引脚上驱动。对于一个总线周期，PWT 和 PCD 位根据周期运行的类型不是从 CR3，PDPE，PDE 就是从 PTE 处获得。

若禁止分页(CR0. PG＝0)并且 CR4. PAE＝0，P5 以与 INTEL 486CPU 相同的方法处理 PCD/PWT 引脚和寄存器位。具体地说，使 PWT 引脚为低电平，PCD 引脚反映 CR0 中的 CD 位。在超高速缓存进程期间忽略 CR3 中的 PCD 和 PWT 位(假设为 0)。

为了避免在允许物理地址扩展(CR4. PAE＝1)时产生的每次 TLB 未命中时的三级查找，就要在重新装入 CR3 时由处理器读入所有 4 个页面目录项。不管 CR0. PE 和 CR0. PG 的状态如何，PCD 和 PWT 引脚反映有关这些参考值的 CR3 值。注意 CR3. PCD 和 CR3. PWT 的

值仍然可由 CR0.CD 超越。

当允许分页和超高速缓存时(CR0 中 PG=1,CD=0),来自最后一项(根据 4M 字节、2M 字节或 4K 字节方式,可能是 PDE 或是 PTE)的若干位超高速缓存在转换后援缓冲器(TLB)中,并在引用由 TLB 项目映象的页面任何时间加以驱动。TLB 更新周期期间读出页面目录和页表项目时,必须在别处获得 PWT 和 PCD 位。下表 10-13 至表 10-16 列出了在 TLB 访问周期期间从何处获得所有寻址和页面大小方式需要的 PWT 和 PCD 位。

表 10-13

36 位/4K 字节页面	
PCD/PWT 取自:	访问期间至:
CR3	PDPE
PDPE	PDE
PDE	PTE
PTE	所有其它分页的存储器访问

表 10-14

36 位/2M 字节页面	
PCD/PWT 取自:	访问期间至:
CR3	PDPE
PDPE	PDE
PDE	所有其它分页的存储器访问

表 10-15

32 位/4K 字节页面	
PCD/PWT 取自:	访问期间至:
CR3	PDE
PDE	PTE
PTE	所有其它分页的存储器访问

表 10-16

32 位/4M 字节页面	
PCD/PWT 取自:	访问期间至:
CR3	PDE
PTE	所有其它分页的存储器访问

若禁止超高速缓存(CR0.CD=1),PCD 引脚则始终为高电平(1)。

对 I/O,中断确认和特定周期来说,PCD 和 PWT 引脚没有任何含义。

注意:改变页面表中的 PCD 和 PWT 不会修改早已在超高速缓存中的许多行的状态位。将受到影响的只是未来的一些行。

图 10-52 所示说明如何在 36 位寻址方式中产生 PCD 的实例。

D/C#,W/R#和 M/IO#三个引脚定义总线周期类型。总线周期类型见表 10-17。

表 10-17 ADS 初始的总线周期定义

M/IO#	D/C#	W/R#	初始的总线周期
0	0	0	中断确认
0	0	1	特定周期
0	1	0	I/O 读
0	1	1	I/O 写
1	0	0	代码读
1	0	1	Intel 公司保留
1	1	0	存储器读
1	1	1	存储器写

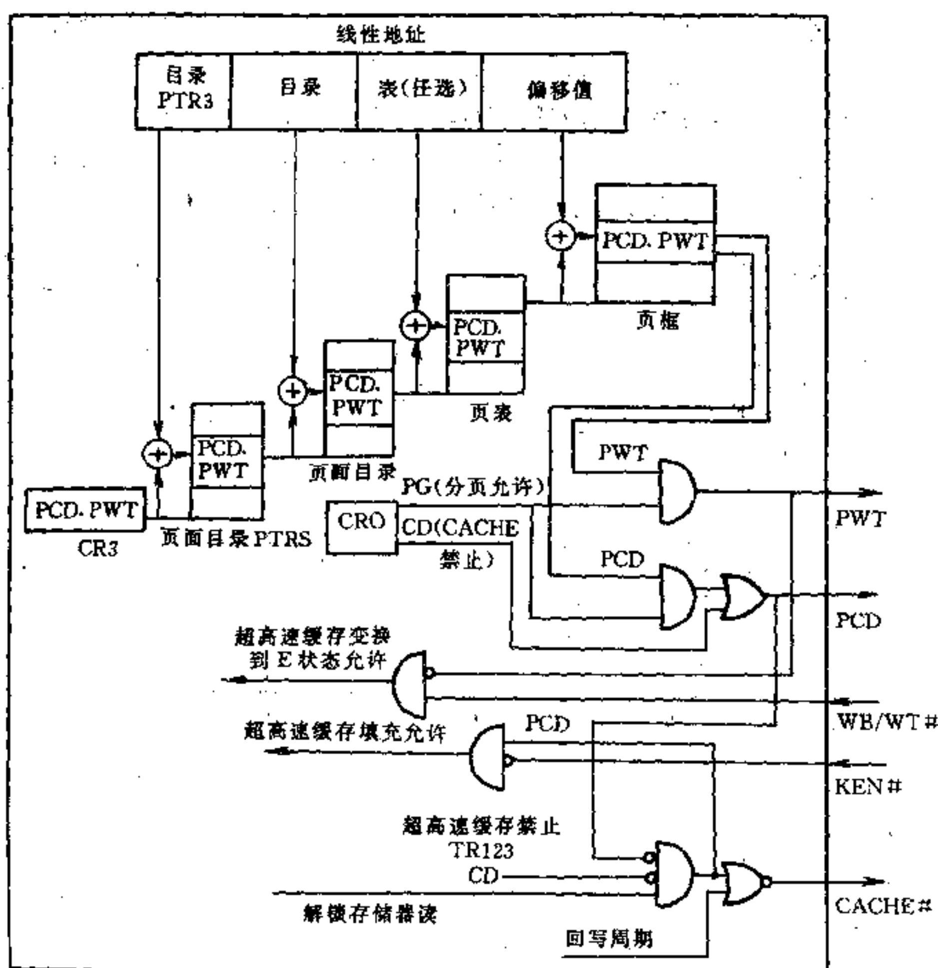


图 10-52 页面超高速缓存功能结构图

P5 提供一个指示超高速缓存能力的 CACHE#引脚。这个引脚作为周期说明部分加以驱动。读周期时建立 CACHE#以指出,如果 KEN#输入恢复活动,该周期将转换成脉冲串行填充。写周期时建立 CACHE#以指出一行回写,指示一次 4 传脉冲串写。

表 10-18 列出了根据 W/R#,CACHE#和 KEN#引脚计算出的不同的周期长度。该表列出了将由 P5 启动的所有数据传送。例如,不可超高速缓存的代码预取周期不能突发。

表 10-18 周期长度

W/R#	CACHE#	KEN#	周期说明
0	1	x	不可超高速缓存的 64 位(或以下)读
0	x	1	不可超高速缓存的 64 位(或以下)读
0	0	0	突发串行填充,256 位读
1	1	x	64 位(或以下)写
1	0	x	突发串回写,256 位写

P5 提供的 LOCK#引脚指示正在运行总线的粒元顺序。这可用来实现以存储器为基础的信号灯。LOCK#将从第一个周期的开始直至最后一个周期 BRDY#被恢复都保持活动(与地

址和其他总线定义引脚一起保持活动)。要保证撤销建立 LOCK#时在相邻的封锁周期之间至少有一个空闲时钟。

如果总线周期在封锁操作期间是分离周期,那么 SCYC 输出引脚为高电平(分离周期是未对准存储器操作数造成的。P5 的硬件认为 2 或 4 字节操作数通过未对准的 4 字节边界。同样,8 字节操作数通过未对准的 8 字节边界。)SCYC 只保证封锁操作时有效,对解锁周期未定义。该引脚同 LOCK#引脚一起可由第二级超高速缓存控制器在供分离周期封锁用时强制总线封锁而不是地址封锁。总线封锁称作读改写操作,它通过封锁到存储器的通路并有效拒绝任何其他总线的事务处理来执行。地址封锁称作第二级超高速缓存中执行的读改写操作。这允许系统中存在多重同时封锁。SCYC 的定时与 M/IO#、D/C#和 W/R#相同。

地址选通输出 ADS#通知外部系统在地址总线上存在一个新地址。无论何时总空闲,这个信号将在总线周期的第一个时钟里保持整个时钟活动,并在总线周期的第二个时钟里保持非活动。注意,当 BOFF#在与 ADS#同一时钟里被建立时,ADS#可以浮动到低电平。因此总线周期的开头应按建立的 ADS#定义,而且在前一时钟中不应建立 BOFF#。

P5 接收一个突发串就绪输入 BRDY#。读出时,数据在建立 BRDY#时选通处理器。写入时,必须建立 BRDY#,以表示外部逻辑已锁存写入数据。

单个输入引脚 KEN#用来允许将数据存入内部超高速缓存。KEN#引脚在恢复第一个 BRDY#或 NA#有效的时钟里取样。如果 P5 正运行一个可超高速缓存的读周期,如建立 CACHE #所示,并且 KEN#引脚恢复活动,读周期会自动转换成一个 4 传突发串。然后使 P5 接收一整行,并且由存储器系统负责返回一个 4 传送突发串。P5 在执行回写周期时忽略 KEN #引脚。

NA#输入是使 P5 在完成当前总线周期前启动一个新的总线周期。在建立 NA#后的两个时钟,即使外部硬件没有启动当前周期的所有 BRDY#,P5 也可以启动新的地址的周期的技术要求。P5 最多支持两个未完成的总线周期。

WB/WT#引脚用来控制正访问行的MESI 状态。WB/WT#引脚在恢复第一个 BRDY#或 NA#有效的时钟里取样。若该引脚取样为低电平,则这行将作为通写项处理。若内部写命中该行,这将给总线产生一个通写周期。若该引脚取样为高电平,并且在总线上没有显示出 MESI 状态转换之后,写命中该行。该引脚是在读和写期间取样的。

在 EWBE#(外部写缓冲器空)输入引脚为高电平时,它指出 P5 在其外面有未决的通写周期(即主存尚未被通写周期更新)。非活动的(高电平)EWBE#将停止随后写入到任何 E 或 M 状态行。这将保证从 P5 外面看到的写入顺序同软件产生它们的顺序相同。

3. 调试和测试引脚

调试和测试引脚见表 10-19。

表 10-19 调试和测试引脚

类 型	引 脚 名	数量
探针方式	R/S#、PRDY	2
断点/性能监测	PM0/BP0、PM1/BP1、BP3～BP2	4
边界扫描	TCK、TDI、TDO、TMS、TRST#	5

P5 将实现一种已知为探针方式的新的调试方式。用这种方式可以检验和修改 P5 的内部

状态和系统的外部状态。处理器寄存器可以读和写。系统存储器和IO空间也可以读和写。

探针方式由边界扫描和专用引脚联合控制和访问。要建立两个新的寄存器PDR（探针数据寄存器）和PBR(探针断点寄存器)，并为边界扫描的测试访问端口(TAP)和P5的执行器所知。这些执行寄存器一起使P5能执行那些在接口寄存器和P5寄存器或外部系统之间传送数据命令。边界扫描引脚(TCK，TDI，TKO，TMS，TRST#)用于P5与外界传送数据、控制时钟和转换状态。另外，两个新引脚PRDY和R/S#用来控制进入和退出探针方式以及保持同步。

输出引脚PRDY用以表示P5处在探针方式并且TAP已准备接受边界扫描/探针方式命令。输入引脚R/S#用来进入和退出探针方式。R/S#是一个边沿敏感输入引脚。

P5有4个调试寄存器，可用于编程以检验断点是否匹配。这些断点通过4个断点引脚BP3～BP0在外部表示是否匹配。其中两个引脚BP0(PM0/BP0)和BP1(PM1/BP1)供性能监测多路选择用。PM0和PM1引脚用来外部表示与监测所选择事件相关的计数器已增1或溢出。探针方式控制寄存器中的BP1、BP0(性能监测/断点)位决定是否为性能监测或断点指示配置了多路选用引脚。

四、Pentium 的主要特点

1. Pentium 总线与 80486 总线的主要区别

P5总线被设计成类似于INTEL80486 CPU总线，但是为了获取更高性能和提供对多重处理系统的更好支持，它牺牲了完全兼容性。

下面是P5总线与80486 CPU总线的区别：

■P5具有64位数据总线，而INTEL486 CPU支持32位数据总线。P5比INTEL486 CPU(BE3#～BE0#)具有更多的字节允许(BE7#～BE0#)和数据奇偶校验引脚(DP7～DP0)。

■P5有一条36位地址总线，而INTEL486 CPU支持一条32位地址总线。

■P5支持地址流水线(NA#)，以提供多达2个未完成周期。

■P5在NA#的早期或第一个BRDY#取样KEN#引脚。KEN#只被取样一次。INTEL486 CPU在超高速缓存行填充周期的第一个和最后一个RDY#/BRDY#之前的时钟两次取样KEN#。

■突发串长度信息由P5经由和地址一起的CACHE#引脚驱动。INTEL486 CPU则是在运行中用BLAST#引脚控制突发串长度。

■P5一个总线周期产生8字节写入，所以它没有PLOCK#引脚。

■在突发串期间，P5不改变地址和字节允许的低阶位。

■P5在突发串周期时需要运行回写和行填充，并且突发串不能中间终止(没有RDY #或BLAST#引脚)。

■P5支持可超高速缓存的突发串周期。

■P5使用下面一些新引脚，支持回写超高速缓存协议：CACHE#、HIT#、HITM#、INV和WB/WT#。

■P5不支持用BS8#和BS16#实现的动态总线带宽。

■P5不允许使每个时钟都无效，或当P5驱动地址总线时使时钟无效。

■P5保证在连续锁定周期之前有一个空闲周期。

■P5提供了SCYC引脚，它指出锁定操作期间的一个分割周期。

■P5 的不可超高速缓存的指令预取是 8 字节而不是 16 字节。

■P5 有一个 INIT 引脚，可以在维持内部超高速缓存状态和浮点机器状态的同时执行复位功能。

■P5 通过 EWBE＃引脚支持它与外部系统之间的强存储排序。

■P5 支持内部奇偶错误校验，加强了数据奇偶校验和地址奇偶校验。建立了下面引脚以实现这些新特性：APCHK＃、BUSCHK＃、PEN＃、IERR＃和 AP。

■P5 可使用下面引脚进行边界扫描：TDI、TDO、TMS、TRS＃和 TCK。

■P5 使用下面引脚支持探针方式：R/S＃和 PRDY。

■P5 含有 IU、IV 和 IBT 引脚，以及一个用于执行跟踪的转移跟踪信息专用周期。

■P5 使用 FRCMC＃和 IERR＃引脚支持功能冗余校验(FRC)。

■P5 使用下面引脚支持性能监测：BP3、BP2、PM1/BP1 和 PM0/BP0。

■P5 使用 SMI＃输入和 SMIACT＃输出实现系统管理方式。

■在 P5 上，使用 BOFF＃异常结束一个总线周期后，从头重新启动该总线周期。不保留在 BOFF＃之前返回的数据。INTEL486 CPU 保留在 BOFF＃建立之前返回的数据，并在周期被异常结束的点上重新启动该周期。

■FLUSH＃是一个边沿触发的输入。对于每个下降边沿，它被识别一次。它是作为一次中断实现的，因此仅在指令边界上被识别。

2. Pentium 的高速缓存器

P5 为了提高其性能和速度，在其芯片上集成了两片 8K 字节的高速缓存器。并在 INTEL80486 CPU 超高速缓存的基础上增加了若干功能。它具有独立的指令和数据超高速缓存以及独立的指令和数据 TLB。数据超高速缓存是一种回写超高速缓存，在多处理器环境下，由一种称为 MESI 协议的超高速一致性来保持数据的一致性。

图 10-53 给出了数据和指令超高速缓存结构的概念图。

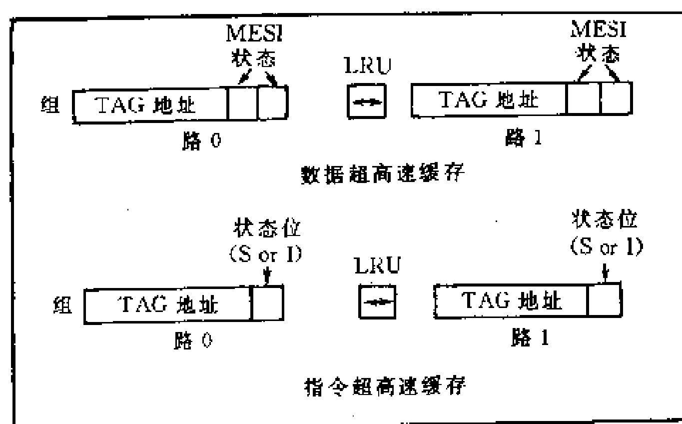


图 10-53 指令和数据超高速缓存的概念结构

3. Pentium 对 80486 指令系统的扩充：

P5 的指令系统包括 80486 的全部指令，并有以下指令的扩充：

■比较和交换 8 字节数据值，以供在多处理器系统中双连接表和其它数据结构的基本操作用。

■读处理器模型和修正级的指令。

■读 64 位时间标记的指令。

■读和写新控制寄存器 CR4 的指令。

■在通用寄存器和模型专用寄存器之间传送数值的指令。

■从系统管理方式返回的指令。

4. Pentium 的探针方式

P5 实现了一种探针方式。在这种新的方式下，可以检查和修改 CPU 内部状态和系统外部状态，可以读和写处理器寄存器，还可以读和写系统存储器和 I/O 空间。

在探针方式下，中断处理器取和执行指令的正常执行序列。CPU 进入待用状态，在该状态绕过正常的预取和译码机制，并可以将指令直接强制入 CPU 的执行部件。指令和时间通过 JTAG 引脚至特定的专用寄存器馈给处理器。这些指令可以读和写寄存器，并且可以读和写存储器或 I/O 空间。它们允许 CPU 执行那些在 CPU 和接口寄存器、CPU 寄存器或外部系统间传输数据的命令。

该方式由边界扫描和专用引脚联合进行控制和访问。P5 使用两个新的引脚 R/S＃和 PRDY 控制探针方式的执行。使用 JTAG 引脚读和写驻留在边界扫描测试访问端口（TAP ）中的探针方式寄存器。这些特性共同对 CPU 的执行部件直接进行外部控制。

（1）基本操作

在探针方式下，处理器将接收简单的装入存储指令。这些指令是微代码的子集，不要经过译码直接送入执行部件。探针方式指令允许读或写所有程序员可视的寄存器，还可以访问存储器和 I/O 空间。在探针方式下会发生窥视和回写。外部中断(SMI，NMI ，Machine ，Check，Flush＃，INIT)将保持待处理状态并在 CPU 退出探针方式时再进行处理。在探针方式下可以识别 RESET，并将导致退出探针方式。在探针方式下忽略 INTR 的建立。

在 CPU 和外部系统之间使用若干寄存器进行通信。探针指令寄存器(PIR)是一种含有“操作码”、源和目的寄存器字段的固定格式指令寄存器。探针数据寄存器(PDR)用来和 CPU 交换数据值。例如，为了读一个寄存器，你必须发送一条将该寄存器内容写入 PDR 的指令，然后再扫描出 PDR。同样地，通过从存储器读入一个将被写入 PDR 的暂时寄存器可以检查存储器。第三个寄存器(探针方式控制器)可以用来作为调试异常的结果进入探针方式而不是调用调试处理程序。

P5 使用 JTAG 接口控制探针方式。正如在 IEEE 标准 1149.1 中所描述的，JTAG 接口定义了一种标准，它允许在组件和外部系统间交换测试数据，并允许向组件提供测试指令。TDI，TDO，TMS，TCK 和 TEST＃引脚在外部系统和片上测试逻辑之间提供了一个串行接口。这种测试逻辑执行一些简单的指令，并含有一些测试寄存器。它称为测试访问端口或 TAP。

（2）进入和退出探针方式

P5 上有两个新的引脚，它们控制进入和退出探针方式并与外部系统同步。这两个引脚是：R/S＃和 PRDY 引脚。R/S＃是一个对边缘触发敏感的输入引脚。PRDY 是一个输出引脚。驱动 R/S＃为低电平将使 CPU 在下一条指令边界停止执行并进入一种空闲状态等待接收探针方式命令（如果没有更高的优先级中断等待处理）。P5 建立 PRDY 引脚，以指出它准备接收探针方式指令。一旦当 CPU 开始执行探针方式指令，PRDY 引脚将进入非活动状态。当 CPU 执行完当前指令而准备接受一条新指令时，PRDY 引脚再次进入活动状态。尽管对在探针方式下可以执行的指令数目没有限制，但要注意所有中断(包括 NMI)都被禁止了。R/S＃从低电平到高电平时使 CPU 退出探针方式。

调试异常也可能导致进入探针方式。当允许该特性时，处理器将进入探针方式而不是调用调试处理程序。通过将探针方式控制寄存器中的中断重新定向位置 1 可以允许这个特性。所有（中断 1）调试异常都被转向探针方式。这包括段点故障、数据段点自陷、单步自陷、任务切换自陷和通用检测故障。（通用检测故障由 DR6 中的第 13 位和 BD 位指出，并通过设置 DR7 中的第 13 位和 GD 位为 1 来允许该故障）。如果由于调试异常而导致进入探针方式，那么在退出时必须强迫 R/S# 为低电平，然后转换为高电平（第一次转换可忽略不计）或者当 R/S# 为高电平时执行退出探针方式 TAP 命令。

通过执行进入探针方式和退出探针方式 TAP 指令也可以进入和退出探针方式。提供 TAP 指令"退出探针方式"是为了退出探针方式并恢复按处理器当前所定义的方式执行。对 R/S # 来说，退出探针方式时它必须是处于非活动状态。如果将 R/S# 拉到低电平而进入探针方式，那么将该引脚拉到高电平就足以恢复执行。

思考题与习题

1. 填空题

(1) 80286 具有多任务系统所必需的（　　）功能、（　　）功能以及（　　）功能，它是能够以多任务方式运行 8086 应用程序的微处理器。

(2) 80286 有（　　）封装和（　　）封装两种形式。两种封装都具有（　　）条引脚，实际用其中的（　　）条。

(3) 80286 有（　　）条地址信号线，其中从（　　）的地址线，输出 I/O 的地址信号，因此访存空间为（　　）字节，而 I/O 空间为（　　）字节。

(4) 80286 的（　　）信号称为系统时钟。在系统内部将系统时钟（　　）分频而形成处理器时钟（PCLK）信号。处理器以 PCLK 同步进行。80286 以（　　）个处理器时钟来执行一次总线周期。总线周期中的第一个处理器时钟称为（　　）周期，以 TS 表示。而第二个处理器周期称为（　　）周期，以 TC 表示。

(5)0286 的内部由（　　　　　　　　）四部分组成。

(6) 80286 段寄存器的种类和 8086 相同，但 80286 的各段寄存器都带有（　　）位的段高速缓存器。它是用来记录（　　），是 80286 进行存储器管理时内部使用的寄存器，这个段高速缓存器的值，不能由外部直接加载。

(7) 80286 比 8086 的标志寄存器多定义了二个标志位，即（　　）和（　　）只能在保护模式下使用，在实模式下不具有意义。

(8) 80286 中新增设了一个 16 位寄存器，只使用低四位，称它为（　　）。

(9) 80286 的段寄存器包含有（　　）位的访问权字段，（　　）位的基地址字段和（　　）位的边界字段。

(10)（　　）字段是程序不能访问的字段，在保护模式下自动地把代码/数据段描述符描述的有效信息加载到这一部分。

(11) 80286 的实模式和保护模式的控制由（　　）的（　　）位决定。

(12) 80286 中表示存储器空间内段位置的地址称为（　　）。

(13) 80286 中段选择器和偏移量称为（　　）。

(14) 系统表寄存器用来在保护模式下管理（　　）个系统表，它们是（　　）。

(15) 80286 的指令系统按照特权级可分为(　　　)等三类。

2. 80286 系统控制指令的功能是什么？写出这些指令的助记符。

3. 何谓 80286 特权指令？写出这些指令的助记符。

4. 简述描述符的分类。

5. 80286 虚拟地址空间的大小是如何确定的？

6. 何谓 80286 的全局地址空间和局部地址空间？

7. 80286 的中断类型可分为几大类？

8. 列表说明 80286 的内部中断。

9. 选择题

(1) 80386 共有(　　)个引脚信号，采用(　　)封装。

A. 128,PGA　　B. 132,LCC

C. 132,PGA　　D. 128,LCC

(2) 80386 内部可以分为(　　)个部件。

A. 8　B. 6　C. 4　D. 2

(3) 80386 有(　　)条地址信号线和(　　)条数据信号线。

A. 32,32　　B. 32,24

C. 24,32　　D. 20,32 2

(4) CLK2 的频率是 80386 内部时钟信号频率的(　　)倍。

A. 5　B. 4　C. 3　D. 2

(5) 80386 的 HLDA 信号是对(　　)的应答信号。

A. HOLD　　B. BUSRQ

C. INTR　　D. DMARQ

(6) 80386 的流水线方式请求信号是(　　)。

A. ND#　　B. NC#

C. NB#　　D. NA#

(7) 80386 的执行部件可分为(　　)子部件。

A. 控制部件　　B. 数据部件

C. 保护测试部件　　D. A 和 B

E. B 和 C　　F. A、B、C

(8) 80386 在保护模式下工作时，可以有(　　)个特权级。

A. 5　B. 4　C. 3　D. 2

10. 80386 的内部有哪些部件？其功能如何？

11. 80386 CPU 有几种操作方式？

12. 80386 CPU 的保护方式提供了哪些保护机制？

13. 80386 的总线操作共分为几种？

14. 简述 80386 CPU 的实地址方式。

15. 简述 80386 CPU 实地址方式的软件初始化。

16. 简述 80386 CPU 虚拟 8086 方式。

17. 如何进入和退出虚拟 8086 方式？

18. 简述 80386 CPU 保护方式的软件初始化。

19. 简述 80386 CPU 进行任务切换的过程。
20. 简述 80386 指令系统的主要特点。
21. 80386 CPU 的指令可分为哪几种类型?
22. 80486 与 80386 的主要区别在哪些地方?
23. 简述 80486 的特点。
24. 简述 80386/80486 的存储器管理。
25. 简述 80386/80486 的分页功能。
26. 80486 浮点指令是与哪些寄存器有关的指令?
27. 简述 80486 片内高速缓存结构。
28. 80386/80486 中有几种高级语言指令?其功能如何?
29. 80486 中新增加的指令有哪几种?其功能如何?
30. 80486 与 80386 在寄存器一级中有哪些不同?

附录A　8086/8088指令对标志位的影响

一、对状态标志的影响

指令类型	指令	OF	SF	ZF	AF	PF	CF
加法，减法，搜索	ADD，ADC，SUB，SBB，SCAS	+	+	+	+	+	+
字符串比较	CMP，NEG，CMPS	+	+	+	+	+	+
增量，减量	INC，DEC	+	+	+	+	+	*
乘法	MUL，IMUL	+	*	*	*	*	+
除法	DIV，IDIV	*	*	*	*	*	*
十进制调整	DAA，DAS	*	+	+	+	+	+
	AAA，AAS	*	*	*	+	*	+
	AAM，AAD	*	+	+	*	+	*
逻辑运算	AND，OR，XOR，TEST	0	+	+	*	+	0
移位	SHL，SHR，SAL，SAR(移一位)	+	+	+	*	+	+
	SHL，SHR，SAL，SAR(移多位)	*	+	+	*	+	+
循环移位	ROL，ROR，RCL，RCR(移位一次)	+					+
	ROL，ROR，RCL，RCR(移位多次)	*					+
设置进位标志	STC						1
	CLC						0
	CMC						CF

二、对控制标志位的影响

指令类型	指令	DF	IF	TF
恢复状态标志	POPF，IRET	+	+	+
中断	INT，INTO		0	0
设置方向标志	STD	1		
	CLD	0		
设置中断标志	STI		1	
	CLI		0	

符号说明：+——有影响；*——不定值；空白——不影响；

0——置0；1——置1

附录B 8086/8088指令编码一览表

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
00	0000 0000	mod reg r/m	(disp-lo),(disp-hi)	ADD reg8/mem8,reg8
01	0000 0001	mod reg r/m	(disp-lo),(disp-hi)	ADD reg16/mem16,reg16
02	0000 0010	mod reg r/m	(disp-lo),(disp-hi)	ADD reg8,reg8/mem8
03	0000 0011	mod reg r/m	(disp-lo),(disp-hi)	ADD reg16,reg16/mem16
04	0000 0100	data-8		ADD AL,data8
05	0000 0101	data-lo	data-hi	ADD AX,data16
06	0000 0110			PUSH ES
07	0000 0111			POP ES
08	0000 1000	mod reg r/m	(disp-lo),(disp-hi)	OR reg8/mem8,reg8
09	0000 1001	mod reg r/m	(disp-lo),(disp-hi)	OR reg16/mem16,reg16
0A	0000 1010	mod reg r/m	(disp-lo),(disp-hi)	OR reg8,reg8/mem8
0B	0000 1011	mod reg r/m	(disp-lo),(disp-hi)	OR reg16,reg16/mem16
0C	0000 1100	data-8		OR AL,data8
0D	0000 1101	data-lo	data-hi	OR AX,data16
0E	0000 1110			PUSH CS
0F	0000 1111			(not used)
10	0001 0000	mod reg r/m	(disp-lo),(disp-hi)	ADC reg8/mem8,reg8
11	0001 0001	mod reg r/m	(disp-lo),(disp-hi)	ADC reg16/mem16,reg16
12	0001 0010	mod reg r/m	(disp-lo),(disp-hi)	ADC reg8,reg8/mem8
13	0001 0011	mod reg r/m	(disp-lo),(disp-hi)	ADC reg16,reg16/mem16
14	0001 0100	data-8		ADC AL,data8
15	0001 0101	data-lo	data-hi	ADC AX,data16
16	0001 0110			PUSH SS
17	0001 0111			POP SS
18	0001 1000	mod reg r/m	(disp-lo),(disp-hi)	SBB reg8/mem8,reg8
19	0001 1001	mod reg r/m	(disp-lo),(disp-hi)	SBB reg16/mem16,reg16
1A	0001 1010	mod reg r/m	(disp-lo),(disp-hi)	SBB reg8,reg8/mem8
1B	0001 1011	mod reg r/m	(disp-lo),(disp-hi)	SBB reg16,reg16/mem16
1C	0001 1100	data-8		SBB AL,data8
1D	0001 1101	data-lo	data-hi	SBB AX,data16
1E	0001 1110			PUSH DS
1F	0001 1111			POP DS
20	0010 0000	mod reg r/m	(disp-lo),(disp-hi)	AND reg8/mem8,reg8
21	0010 0001	mod reg r/m	(disp-lo),(disp-hi)	AND reg16/mem16,reg16

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
22	0010 0010	mod reg r/m	(disp-lo),(disp-hi)	AND reg8,reg8/mem8
23	0010 0011	mod reg r/m	(disp-lo),(disp-hi)	AND reg16,reg16/mem16
24	0010 0100	data-8		AND AL,data8
25	0010 0101	data-lo	data-hi	AND AX,data16
26	0010 0110			ES:(segment override prefix)
27	0010 0111			DAA
28	0010 1000	mod reg r/m	(disp-lo),(disp-hi)	SUB reg8/mem8,reg8
29	0010 1001	mod reg r/m	(disp-lo),(disp-hi)	SUB reg16/mem16,reg16
2A	0010 1010	mod reg r/m	(disp-lo),(disp-hi)	SUB reg8,reg8/mem8
2B	0010 1011	mod reg r/m	(disp-lo),(disp-hi)	SUB reg16,reg16/mem16
2C	0010 1100	data-8		SUB AL,data8
2D	0010 1101	data-lo	data-hi	SUB AX,data16
2E	0010 1110			CS:(segment override prefix)
2F	0010 1111			DAS
30	0011 0000	mod reg r/m	(disp-lo),(disp-hi)	XOR reg8/mem8,reg8
31	0011 0001	mod reg r/m	(disp-lo),(disp-hi)	XOR reg16/mem16,reg16
32	0011 0010	mod reg r/m	(disp-lo),(disp-hi)	XOR reg8,reg8/mem8
33	0011 0011	mod reg r/m	(disp-lo),(disp-hi)	XOR reg16,reg16/mem16
34	0011 0100	data-8		XOR AL,data8
35	0011 0101	data-lo	data-hi	XOR AX,data16
36	0011 0110			SS:(segment override prefix)
37	0011 0111			AAA
38	0011 1000	mod reg r/m	(disp-lo),(disp-hi)	CMP reg8/mem8,reg8
39	0011 1001	mod reg r/m	(disp-lo),(disp-hi)	CMP reg16/mem16,reg16
3A	0011 1010	mod reg r/m	(disp-lo),(disp-hi)	CMP reg8,reg8/mem8
3B	0011 1011	mod reg r/m	(disp-lo),(disp-hi)	CMP reg16,reg16/mem16
3C	0011 1100	data-8		CMP AL,data8
3D	0011 1101	data-lo	data-hi	CMP AX,data16
3E	0011 1110			DS:(segment override prefix)
3F	0011 1111			AAS
40	0100 0000			INC AX
41	0100 0001			INC CX
42	0100 0010			INC DX
43	0100 0011			INC BX
44	0100 0100			INC SP

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
45	0100 0101			INC BP
46	0100 0110			INC SI
47	0100 0111			INC DI
48	0100 1000			DEC AX
49	0100 1001			DEC CX
4A	0100 1010			DEC DX
4B	0100 1011			DEC BX
4C	0100 1100			DEC SP
4D	0100 1101			DEC BP
4E	0100 1110			DEC SI
4F	0100 1111			DEC DI
50	0101 0000			PUSH AX
51	0101 0001			PUSH CX
52	0101 0010			PUSH DX
53	0101 0011			PUSH BX
54	0101 0100			PUSH SP
55	0101 0101			PUSH BP
56	0101 0110			PUSH SI
57	0101 0111			PUSH DI
58	0101 1000			POP AX
59	0101 1001			POP CX
5A	0101 1010			POP DX
5B	0101 1011			POP BX
5C	0101 1100			POP SP
5D	0101 1101			POP BP
5E	0101 1110			POP SI
5F	0101 1111			POP DI
60	0110 0000			(not used)
61	0110 0001			(not used)
62	0110 0010			(not used)
63	0110 0011			(not used)
64	0110 0100			(not used)
65	0110 0101			(not used)
66	0110 0110			(not used)
67	0110 0111			(not used)
68	0110 1000			(not used)

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
69	0110 1001			(not used)
6A	0110 1010			(not used)
6B	0110 1011			(not used)
6C	0110 1100			(not used)
6D	0110 1101			(not used)
6E	0110 1110			(not used)
6F	0110 1111			(not used)
70	0111 0000	IP-INC8		JO SHORT-LABEL
71	0111 0001	IP-INC8		JNO SHORT-LABEL
72	0111 0010	IP-INC8		JB/JNAE/JC SHORT-LABEL
73	0111 0011	IP-INC8		JNB/JAE/JNC SHORT-LABEL
74	0111 0100	IP-INC8		JE/JZ SHORT-LABEL
75	0111 0101	IP-INC8		JNE/JNZ SHORT-LABEL
76	0111 0110	IP-INC8		JBE/JNA SHORT-LABEL
77	0111 0111	IP-INC8		JNBE/JA SHORT-LABEL
78	0111 1000	IP-INC8		JS
79	0111 1001	IP-INC8		JNS
7A	0111 1010	IP-INC8		JP/JPE SHORT-LABEL
7B	0111 1011	IP-INC8		JNP/JPO SHORT-LABEL
7C	0111 1100	IP-INC8		JL/JNGE SHORT-LABEL
7D	0111 1101	IP-INC8		JNL/JGE SHORT-LABEL
7E	0111 1110	IP-INC8		JLE/JNG SHORT-LABEL
7F	0111 1111	IP-INC8		JNLE/JG SHORT-LABEL
80	1000 0000	mod 000 r/m	(disp-lo),(disp-hi) data-8	ADD reg8/mem8,data8
80	1000 0000	mod 001 r/m	(disp-lo),(disp-hi) data-8	OR reg8/mem8,data8
80	1000 0000	mod 010 r/m	(disp-lo),(disp-hi) data-8	ADC reg8/mem8,data8
80	1000 0000	mod 011 r/m	(disp-lo),(disp-hi) data-8	SBB reg8/mem8,data8
80	1000 0000	mod 100 r/m	(disp-lo),(disp-hi) data-8	AND reg8/mem8,data8
80	1000 0000	mod 101 r/m	(disp-lo),(disp-hi) data-8	SUB reg8/mem8,data8
80	1000 0000	mod 110 r/m	(disp-lo),(disp-hi)	XOR reg8/mem8,data8

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
			data-8	
80	1000 0000	mod 111 r/m	(disp-lo),(disp-hi) data-8	CMP reg8/mem8,data8
81	1000 0001	mod 000 r/m	(disp-lo),(disp-hi) data-lo,data-hi	ADD reg16/mem16,data16
81	1000 0001	mod 001 r/m	(disp-lo),(disp-hi) data-lo,data-hi	OR reg16/mem16,data16
81	1000 0001	mod 010 r/m	(disp-lo),(disp-hi) data-lo,data-hi	ADC reg16/mem16,data16
81	1000 0001	mod 011 r/m	(disp-lo),(disp-hi) data-lo,data-hi	SBB reg16/mem16,data16
81	1000 0001	mod 100 r/m	(disp-lo),(disp-hi) data-lo,data-hi	AND reg16/mem16,data16
81	1000 0001	mod 101 r/m	(disp-lo),(disp-hi) data-lo,data-hi	SUB reg16/mem16,data16
81	1000 0001	mod 110 r/m	(disp-lo),(disp-hi) data-lo,data-hi	XOR reg16/mem16,data16
81	1000 0001	mod 111 r/m	(disp-lo),(disp-hi) data-lo,data-hi	CMP reg16/mem16,data16
82	1000 0010	mod 000 r/m	(disp-lo),(disp-hi) data-8	ADD reg8/mem8,data8
82	1000 0010	mod 001 r/m		(not used)
82	1000 0010	mod 010 r/m	(disp-lo),(disp-hi) data-8	ADC reg8/mem8,data8
82	1000 0010	mod 011 r/m	(disp-lo),(disp-hi) data-8	SBB reg8/mem8,data8
82	1000 0010	mod 100 r/m		(not used)
82	1000 0010	mod 101 r/m	(disp-lo),(disp-hi) data-8	SUB reg8/mem8,data8
82	1000 0010	mod 110 r/m		(not used)
82	1000 0010	mod 111 r/m	(disp-lo),(disp-hi) data-8	CMP reg8/mem8,data8
83	1000 0011	mod 000 r/m	(disp-lo),(disp-hi) data-lo,data-hi	ADD reg16/mem16,data16
83	1000 0011	mod 001 r/m		(not used)
83	1000 0011	mod 010 r/m	(disp-lo),(disp-hi)	ADC reg16/mem16,data16

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
			data-lo,data-hi	
83	1000 0011	mod 011 r/m	(disp-lo),(disp-hi)	SBB reg16/mem16,data16
			data-lo,data-hi	
83	1000 0011	mod 100 r/m		(not used)
83	1000 0011	mod 101 r/m	(disp-lo),(disp-hi)	SUB reg16/mem16,data16
			data-lo,data-hi	
83	1000 0011	mod 110 r/m		(not used)
83	1000 0011	mod 111 r/m	(disp-lo),(disp-hi)	CMP reg16/mem16,data16
			data-lo,data-hi	
84	1000 0100	mod reg r/m	(disp-lo),(disp-hi)	TEST reg8/mem8,reg8
85	1000 0101	mod reg r/m	(disp-lo),(disp-hi)	TEST reg16/mem16,reg16
86	1000 0110	mod reg r/m	(disp-lo),(disp-hi)	XCHG reg8,reg8/mem8
87	1000 0111	mod reg r/m	(disp-lo),(disp-hi)	XCHG reg16,reg16/mem16
88	1000 1000	mod reg r/m	(disp-lo),(disp-hi)	MOV reg8/mem8,reg8
89	1000 1001	mod reg r/m	(disp-lo),(disp-hi)	MOV reg16/mem16,reg16
8A	1000 1010	mod reg r/m	(disp-lo),(disp-hi)	MOV reg8,reg8/mem8
8B	1000 1011	mod reg r/m	(disp-lo),(disp-hi)	MOV reg16,reg16/mem16
8C	1000 1100	mod 0sr r/m	(disp-lo),(disp-hi)	MOV reg16/mem16,segreg
8C	1000 1100	mod 1- r/m		(not used)
8D	1000 1101	mod reg r/m	(disp-lo),(disp-hi)	LEA reg16,mem16
8E	1000 1110	mod 0sr r/m	(disp-lo),(disp-hi)	MOV segreg,reg16/mem16
8E	1000 1110	mod 1- r/m		(not used)
8F	1000 1111	mod 000 r/m		POP reg16/mem16
8F	1000 1111	mod 001 r/m		(not used)
8F	1000 1111	mod 010 r/m		(not used)
8F	1000 1111	mod 011 r/m		(not used)
8F	1000 1111	mod 100 r/m		(not used)
8F	1000 1111	mod 101 r/m		(not used)
8F	1000 1111	mod 110 r/m		(not used)
8F	1000 1111	mod 111 r/m		(not used)
90	1001 0000			NOP
91	1001 0001			XCHG AX,CX
92	1001 0010			XCHG AX,DX
93	1001 0011			XCHG AX,BX
94	1001 0100			XCHG AX,SP
95	1001 0101			XCHG AX,BP

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
96	1001 0110			XCHG AX,SI
97	1001 0111			XCHG AX,DI
98	1001 1000			CBW
99	1001 1001			CWD
9A	1001 1010	disp-lo	disp-hi,seg-lo,seg-hi	CALL FAR PROC
9B	1001 1011			WAIT
9C	1001 1100			PUSHF
9D	1001 1101			POPF
9E	1001 1110			SAHF
9F	1001 1111			LAHF
A0	1010 0000	addr-lo	addr-hi	MOV AL,mem8
A1	1010 0001	addr-lo	addr-hi	MOV AX,mem16
A2	1010 0010	addr-lo	addr-hi	MOV mem8,AL
A3	1010 0011	addr-lo	addr-hi	MOV mem16,AX
A4	1010 0100			MOVS dst-str8,src-str8
A5	1010 0101			MOVS dst-str16,src-str16
A6	1010 0110			CMPS dst-str8,src-str8
A7	1010 0111			CMPS dst-str16,src-str16
A8	1010 1000	data-8		TEST AL,data8
A9	1010 1001	data-lo	addr-hi	TEST AX,data16
AA	1010 1010			STOS dst-string8
AB	1010 1011			STOS dst-string16
AC	1010 1100			LODS src-string8
AD	1010 1101			LODS src-string16
AE	1010 1110			SCAS dst-string8
AF	1010 1111			SCAS dst-string16
B0	1011 0000	data-8		MOV AL,data8
B1	1011 0001	data-8		MOV CL,data8
B2	1011 0010	data-8		MOV DL,data8
B3	1011 0011	data-8		MOV BL,data8
B4	1011 0100	data-8		MOV AH,data8
B5	1011 0101	data-8		MOV CH,data8
B6	1011 0110	data-8		MOV DH,data8
B7	1011 0111	data-8		MOV BH,data8
B8	1011 1000	data-lo	data-hi	MOV AX,data16
B9	1011 1001	data-lo	data-hi	MOV CX,data16

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
BA	1011 1010	data-lo	data-hi	MOV DX,data16
BB	1011 1011	data-lo	data-hi	MOV BX,data16
BC	1011 1100	data-lo	data-hi	MOV SP,data16
BD	1011 1101	data-lo	data-hi	MOV BP,data16
BE	1011 1110	data-lo	data-hi	MOV SI,data16
BF	1011 1111	data-lo	data-hi	MOV DI,data16
C0	1100 0000			(not used)
C1	1100 0001			(not used)
C2	1100 0010	data-lo	data-hi	RET data16(intraseg)
C3	1100 0011			RET (intrasegment)
C4	1100 0100	mod reg r/m	(disp-lo),(disp-hi)	LES reg16,mem16
C5	1100 0100	mod reg r/m	(disp-lo),(disp-hi)	LDS reg16,mem16
C6	1100 0101	mod 000 r/m	(disp-lo),(disp-hi)	MOV mem8,data8
			data8	
C6	1100 0110	mod 001 r/m		(not used)
C6	1100 0110	mod 010 r/m		(not used)
C6	1100 0110	mod 011 r/m		(not used)
C6	1100 0110	mod 100 r/m		(not used)
C6	1100 0110	mod 101 r/m		(not used)
C6	1100 0110	mod 110 r/m		(not used)
C6	1100 0110	mod 111 r/m		(not used)
C7	1100 0111	mod 000 r/m	(disp-lo),(disp-hi)	MOV mem16,data16
			data-lo,data-hi	
C7	1100 0111	mod 001 r/m		(not used)
C7	1100 0111	mod 010 r/m		(not used)
C7	1100 0111	mod 011 r/m		(not used)
C7	1100 0111	mod 100 r/m		(not used)
C7	1100 0111	mod 101 r/m		(not used)
C7	1100 0111	mod 110 r/m		(not used)
C7	1100 0111	mod 111 r/m		(not used)
C8	1100 1000			(not used)
C9	1100 1001			(not used)
CA	1100 1010	data-lo	data-hi	RET data16(intersegment)
CB	1100 1011			RET (intersegment)
CC	1100 1100			INT 3
CD	1100 1101	data-8		INT data8

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
CE	1100 1110			INTO
CF	1100 1111			IRET
D0	1101 0000	mod 000 r/m	(disp-lo),(disp-hi)	ROL reg8/mem8,1
D0	1101 0000	mod 001 r/m	(disp-lo),(disp-hi)	ROR reg8/mem8,1
D0	1101 0000	mod 010 r/m	(disp-lo),(disp-hi)	RCL reg8/mem8,1
D0	1101 0000	mod 011 r/m	(disp-lo),(disp-hi)	RCR reg8/mem8,1
D0	1101 0000	mod 100 r/m	(disp-lo),(disp-hi)	SAL/SHL reg8/mem8,1
D0	1101 0000	mod 101 r/m	(disp lo),(disp-hi)	SHR reg8/mem8,1
D0	1101 0000	mod 110 r/m		(not used)
D0	1101 0000	mod 111 r/m	(disp-lo),(disp-hi)	SAR reg8/mem8,1
D1	1101 0001	mod 000 r/m	(disp-lo),(disp-hi)	ROL reg16/mem16,1
D1	1101 0001	mod 001 r/m	(disp-lo),(disp-hi)	ROR reg16/mem16,1
D1	1101 0001	mod 010 r/m	(disp-lo),(disp-hi)	RCL reg16/mem16,1
D1	1101 0001	mod 011 r/m	(disp-lo),(disp-hi)	RCR reg16/mem16,1
D1	1101 0001	mod 100 r/m	(disp-lo),(disp-hi)	SAL/SHL reg16/mem16,1
D1	1101 0001	mod 101 r/m	(disp-lo),(disp-hi)	SHR reg16/mem16,1
D1	1101 0001	mod 110 r/m		(not used)
D1	1101 0001	mod 111 r/m	(disp-lo),(disp-hi)	SAR reg16/mem16,1
D2	1101 0010	mod 000 r/m	(disp-lo),(disp-hi)	ROL reg8/mem8,CL
D2	1101 0010	mod 001 r/m	(disp-lo),(disp-hi)	ROR reg8/mem8,CL
D2	1101 0010	mod 010 r/m	(disp-lo),(disp-hi)	RCL reg8/mem8,CL
D2	1101 0010	mod 011 r/m	(disp-lo),(disp-hi)	RCR reg8/mem8,CL
D2	1101 0010	mod 100 r/m	(disp-lo),(disp-hi)	SAL/SHL reg8/mem8,CL
D2	1101 0010	mod 101 r/m	(disp-lo),(disp-hi)	SHR reg8/mem8,CL
D2	1101 0010	mod 110 r/m		(not used)
D2	1101 0010	mod 111 r/m	(disp-lo),(disp-hi)	SAR reg8/mem8,CL
D3	1101 0011	mod 000 r/m	(disp-lo),(disp-hi)	ROL reg16/mem16,CL
D3	1101 0011	mod 001 r/m	(disp-lo),(disp-hi)	ROR reg16/mem16,CL
D3	1101 0011	mod 010 r/m	(disp-lo),(disp-hi)	RCL reg16/mem16,CL
D3	1101 0011	mod 011 r/m	(disp-lo),(disp-hi)	RCR reg16/mem16,CL
D3	1101 0011	mod 100 r/m	(disp-lo),(disp-hi)	SAL/SHL reg16/mem16,CL
D3	1101 0011	mod 101 r/m	(disp-lo),(disp-hi)	SHR reg16/mem16,CL
D3	1101 0011	mod 110 r/m		(not used)
D3	1101 0011	mod 111 r/m	(disp-lo),(disp-hi)	SAR reg16/mem16,CL
D4	1101 0100	00001010		AAM
D5	1101 0101	00001010		AAD

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
D6	1101 0110			(not used)
D7	1101 0111			XLAT src-table
D8	1101 1000	mod 000 r/m		
D9	1xxx	mod yyy r/m	(disp-lo),(disp-hi)	ESC opcode,src
DF	1101 1111	mod 111 r/m		
E0	1110 0000	ip-inc-8		LOOPNE SHORT-LABEL
				LOOPNZ
E1	1110 0001	ip-inc-8		LOOPE SHORT-LABEL
				LOOPZ
E2	1110 0010	ip-inc-8		LOOP SHORT-LABEL
E3	1110 0011	ip-inc-8		JCXZ SHORT-LABEL
E4	1110 0100	data-8		IN AL,port8
E5	1110 0101	data-8		IN AX,port8
E6	1110 0110	data-8		OUT port8,AL
E7	1110 0111	data-8		OUT port8,AX
E8	1110 1000	ip-inc-lo	ip-inc-hi	CALL NEAR-PROC
E9	1110 1001	ip-inc-lo	ip-inc-hi	JMP NEAR-LABEL
EA	1110 1010	ip-lo	ip-hi,cs-lo,cs-hi	JMP FAR-LABEL
EB	1110 1011	ip-inc8		JMP SHORT-LABEL
EC	1110 1100			IN AL,DX
ED	1110 1101			IN AX,DX
EE	1110 1110			OUT DX,AL
EF	1110 1111			OVT DX,AX
F0	1111 0000			LOCK (prefix)
F1	1111 0001			(not used)
F2	1111 0010			REPNE/REPNZ
F3	1111 0011			REP/REPE/REPZ
F4	1111 0100			HLT
F5	1111 0101			CMC
F6	1111 0110	mod 000 r/m	(disp-lo),(disp-hi)	TEST reg8/mem8,data8
			data-8	
F6	1111 0110	mod 001 r/m		(not used)
F6	1111 0110	mod 010 r/m	(disp-lo),(disp-hi)	NOT reg8/mem8
F6	1111 0110	mod 011 r/m	(disp-lo),(disp-hi)	NEG reg8/mem8
F6	1111 0110	mod 100 r/m	(disp-lo),(disp-hi)	MUL reg8/mem8
F6	1111 0110	mod 101 r/m	(disp-lo),(disp-hi)	IMUL reg8/mem8

第一字节		第二字节	第三、四、五、六字节	ASM-86 指令格式
HEX	BINARY			
F6	1111 0110	mod 110 r/m	(disp-lo),(disp-hi)	DIV reg8/mem8
F6	1111 0110	mod 111 r/m	(disp-lo),(disp-hi)	IDIV reg8/mem8
F7	1111 0111	mod 000 r/m	(disp-lo),(disp-hi)	TEST reg16/mem16/data16
			data-lo,data-hi	
F7	1111 0111	mod 001 r/m		(not used)
F7	1111 0111	mod 010 r/m	(disp-lo),(disp-hi)	NOT reg16/mem16
F7	1111 0111	mod 011 r/m	(disp-lo),(disp-hi)	NEG reg16/mem16
F7	1111 0111	mod 100 r/m	(disp-lo),(disp-hi)	MUL reg16/mem16
F7	1111 0111	mod 101 r/m	(disp-lo),(disp-hi)	IMUL reg16/mem16
F7	1111 0111	mod 110 r/m	(disp-lo),(disp-hi)	DIV reg16/mem16
F7	1111 0111	mod 111 r/m	(disp-lo),(disp-hi)	IDIV reg16/mem16
F8	1110 1000			CLC
F9	1110 1001			STC
FA	1110 1010			CLI
FB	1110 1011			STI
FC	1110 1100			CLD
FD	1110 1101			STD
FE	1111 1110	mod 000 r/m	(disp-lo),(disp-hi)	INC reg8/mem8
FE	1111 1110	mod 001 r/m	(disp-lo),(disp-hi)	DEC reg8/mem8
FE	1111 1110	mod 010 r/m		(not used)
FE	1111 1110	mod 011 r/m		(not used)
FE	1111 1110	mod 100 r/m		(not used)
FE	1111 1110	mod 101 r/m		(not used)
FE	1111 1110	mod 110 r/m		(not used)
FE	1111 1110	mod 111 r/m		(not used)
FF	1111 1111	mod 000 r/m	(disp-lo),(disp-hi)	INC mem16
FF	1111 1111	mod 001 r/m	(disp-lo),(disp-hi)	DEC mem16
FF	1111 1111	mod 010 r/m	(disp-lo),(disp-hi)	CALL reg16/mem16(intra)
FF	1111 1111	mod 011 r/m	(disp-lo),(disp-hi)	CALL mem16(intersegment)
FF	1111 1111	mod 100 r/m	(disp-lo),(disp-hi)	JMP reg16/mem16(intra)
FF	1111 1111	mod 101 r/m	(disp-lo),(disp-hi)	JMP mem16(intersegment)
FF	1111 1111	mod 110 r/m	(disp-lo),(disp-hi)	PUSH mem16
FF	1111 1111	mod 111 r/m		(not used)

注：data-SX:将 8 位立即数符号位扩展为 16 位立即数

( ):可缺省部分

intrasegment:段间

intersegment:段内

IP-INC8:短转移偏移地址

附录C ASCII码控制符号的定义

控制符	说　明		控制符	说　明	
NUL	Null	空白	DLE	Data line escape	转义
SOH	Start of heading	序始	DC1	Device control 1	机控 1
STX	Start of text	文始	DC2	Device control 2	机控 2
ETX	End of text	文终	DC3	Device control 3	机控 3
EOT	End of tape	送毕	DC4	Device control 4	机控 4
ENQ	Enquiry	询问	NAK	Negative acknowledge	未应答
ACK	Acknowledge	应答	SYN	Synchronize	同步
BEL	Bell	响铃	ETB	End of transmitted block	组终
BS	Backspace	退格	CAN	Cancel	作废
HT	Horizontal tab	横表	EM	End of medium	载终
LF	Line feed	换行	SUB	Substitute	取代
VT	Vertical tab	纵表	ESC	Escape	换码
FF	Form feed	换页	FS	File separator	文件隔离符
CR	Carriage return	回车	GS	Group separator	组隔离符
SO	Shift out	移出	RS	Record separator	记录隔离符
SI	Shift in	移入	US	Union separator	单元隔离符
SP	Space	空格	DEL	Delete	删除

附录D　系统功能调用一览表

功能号	功　能	入口信息	出口信息
0	程序结束	(AH)=00H (CS)=程序前缀区段界地址	
1	键盘输入单字符	(AH)=01H	(AL)=输入字符编码并屏幕显示键入字符
2	显示输出单字符	(AH)=02H (DL)=字符编码	显示或打印输出单字符
3	异步通信口输入(传输速度为2400bps)	(AH)=03H	(AL)=通信口传送的字符编码无校验
4	异步通信口输出(传输速度为2400bps)	(AH)=04H (DL)=字符编码	串行输出DL中的字符
5	打印机输出	(AH)=05H (DL)=字符编码	
6	直接控制台输入输出(不检查Break键)	(AH)=06H 若(DL)=0FFH表示输入 若(DL)≠0FFH表示输出 DL中为输出字符编码	当(DL)=0FFH,如果有字符则输入到AL中,否则(AL)=0 当(DL)≠0FFH,则输出DL中的字符
7	无回显直接控制台输入(不做字符检查)	(AH)=07H	(AL)=输入字符编码
8	无回显的键盘输入(做字符检查)	(AH)=08H	(AL)=输入字符编码
9	显示输出字符串	(AH)=09H (DS:DX)指向字符串首址, 要求字符串以'$'结尾	
A	键盘输入字符串	(AH)=0AH (DS:DX)指向缓冲区首址	
B	检查键盘输入状态	(AH)=0BH	(AL)=00H无输入 (AL)=0FFH有输入
C	清键盘缓冲区并执行键盘输入功能	(AH)=0CH (AL)=模块号(1,6,7,8或A)	
D	重置磁盘	(AH)=0DH	
E	确定默认磁盘	(AH)=0EH (DL)=磁盘号	(AL)=系统中盘数
F	打开文件	(AH)=0FH (DS:DX)=FCB首址	(AL)=0FFH不成功 (AL)=0　成功 $FCB_{C,D}$及$FCB_{10\sim 25}$被设置

功能号	功　能	入口信息	出口信息
10	关闭文件	(AH)=10H	(AL)=0FFH 不成功
		(DS:DX)=FCB 首址	(AL)=0　成功
11	查找文件名或	(AH)=11H	(AL)=0FFH 未找到
	查找第一个目录项	(DS:DX)=FCB 首址	(AL)=0　找到
12	查找下一个目录项	(AH)=12H	(AL)=0FFH 未找到
		(DS:DX)=FCB 首址	(AL)=0　找到
13	删除文件	(AH)=13H	(AL)=0FFH 不成功
		(DS:DX)=FCB 首址	(AL)=0　成功
14	顺序读一个记录	(AH)=14H	(AL)=00　成功
		(DS:DX)=FCB 首址	(AL)=01　文件结束
		DTA 缓冲区已设置	(AL)=02　缓冲区不够
			(AL)=03　读部分记录而结束
15	顺序写一个记录	(AH)=15H	(AL)=01　成功
		(DS:DX)=FCB 首址	(AL)=02　盘空间不足
		DTA 缓冲区已设置	(AL)=03　缓冲区空间不足
16	建立文件	(AH)=16H	(AL)=00　成功
	(建立新的或旧的)	(DS:DX)=FCB 首址	(AL)=0FFH 目录区满
17	改文件名	(AH)=17H	(AL)=00　成功
		(DS:DX)=FCB 首址	(AL)=0FFH 不成功
		(DS:DX+17)=新文件名首址	
18	DOS 使用		
19	取当前默认驱动器号	(AH)=19H	(AL)=驱动器号
1A	设置 DTA	(AH)=1AH	
		(DS:DX)=DTA 首址	
1B	取文件分配表(FAT)	(AH)=1BH	(DS:BX)=盘类型字节地址
	的有关信息		(DX)=FAT 表项数
			(AL)=分配单元扇区数
			(CX)=物理扇区字节数
1C	取指定盘文件分配表	(AH)=1CH	(DS:BX)=盘类型字节地址
	(FAT)的有关信息	(DL)=驱动器号	(DX)=FAT 表项数
			(AL)=分配单元扇区数
			(CX)=物理扇区字节数
1D	DOS 内部使用		
1E	DOS 内部使用		
1F	DOS 内部使用		
20	DOS 内部使用		

功能号	功　能	入口信息	出口信息
21	随机读一个记录	(AH)=21H (DS:DX)=FCB 首址 DTA 已设置	(AL)=00　成功 (AL)=01　文件结束 (AL)=02　缓冲区不够 (AL)=03　读部分记录而结束
22	随机写一个记录	(AH)=22H (DS:DX)=FCB 首址 DTA 已设置并填好	(AL)=00　成功 (AL)=01　盘空间不足 (AL)=02　DTA 不够
23	取文件长度	(AH)=23H (DS:DX)=FCB 首址	(AL)=00　成功，长度在 FCB 中 (AL)=0FFH 不成功
24	置随机记录号	(AH)=24H (DS:DX)=FCB 首址	
25	设置中断向量	(AH)=25H (DS:DX)=入口地址 (AL)=中断方式码	
26	建立一个程序段	(AH)=26H (DX)=段号	
27	随机块读出	(AH)=27H (DS:DX)=FCB 首址 (CX)=记录数 DTA 已设置	(AL)=00　成功 (AL)=01　文件结束并读完 (AL)=02　缓冲区不够 (AL)=03　最后为部分记录
28	随机块写入	(AH)=28H (DS:DX)=FCB 首址 (CX)=记录数 DTA 已设置并填好	(AL)=00　成功 (AL)=01　盘空间不够 (AL)=02　DTA 不够
29	建立 FCB	(AH)=29H (ES:DI)=FCB 首址 (DS:SI)=字符串(文件名) (AL)=0E 非法字符检查位	(ES:DI)=格式化后的 FCB 首址 (AL)=00　标准文件 (AL)=01　多义文件 (AL)=0FFH 非法盘标识符
2A	取日期	(AH)=2AH	(CX:DX)=日期
2B	设置日期	(AH)=2BH (CX:DX)=日期	(AL)=00　成功 (AL)=0FFH 失败
2C	取时间	(AH)=2CH	(CX:DX)=时间
2D	设置时间	(AH)=2DH (CX:DX)=时间	(AL)=00　成功 (AL)=0FFH 失败
2E	置写盘校验状态	(AH)=2EH (DL)=0,(AL)=状态	
2F	取 DTA 首址	(AH)=2FH	(ES:BX)=DTA 首址

功能号	功　能	入口信息	出口信息
30	取 DOS 版本号	(AH)=30H	(AL)=版本号 (AH)=发行号
31	结束程序并留在内存	(AH)=31H (AL)=退出码 (DL)=程序长度(按块计算)	
32	DOS 内部使用		
33	Break 检查	(AH)=33H 若(AL)=0 为取状态 若(AL)=1 为置状态 (DL)=状态 状态=$\begin{cases}00 & 表示关\\01 & 表示开\end{cases}$	(DL)=状态
34	DOS 内部使用		
35	取中断向量	(AH)=35H (AL)=中断方式码	(ES:BX)=入口地址
36	取盘自由空间数	(AH)=36H (DL)=驱动器号	若(AX)=0FFFFH 无效驱动器号 否则成功并且 (BX)=可用簇数 (DX)=总簇数 (CX)=扇区字节数 (AX)=每簇扇区数
37	DOS 内部使用		
38	取国别信息	(AH)=38H (DS:DX)=信息区(32 字节)首址 (AL)=0	(DS:DX)=信息区首址 其中有国别信息 (CF)=0　正常 (CF)=1　出错
39	建立子目录	(AH)=39H (DS:DX)=字符串地址	(CF)=0　成功　(AX)=3 (CF)=1　失败　(AX)=5
3A	删除子目录	(AH)=3AH (DS:DX)=字符串地址	(CF)=0　成功 (CF)=1　失败 (AX)=3　找不到路径 (AX)=5　拒绝存取
3B	改变当前目录	(AH)=3BH (DS:DX)=字符串地址	(CF)=0　成功 (CF)=1　失败　(AX)=3
3C	建立文件	(AH)=3CH (DS:DX)=字符串地址 字符串为:驱动器名.路径名	若(CF)=0 成功　则(AX)=文件号 若(CF)=1　失败 则(AX)=3　路径找不到

功能号	功　能	入口信息	出口信息
		文件名.扩展名	(AX)=4　打开文件太多
		(CX)=文件属性[注]	(AX)=5　拒绝存取
3D	打开文件	(AH)=3DH	若(CF)=0　成功　(AX)=文件号
		(DS:DX)=字符串地址	若(CF)=1　失败
		AL 中为存取码	则(AX)=12 无效的存取码
		(AL)=0 读	(AX)=2　文件找不到
		(AL)=1 写	(AX)=3　路径找不到
		(AL)=2 读/写	(AX)=4　打开文件太多
			(AX)=5　拒绝存取
3E	关闭文件	(AH)=3EH	若(CF)=0　成功
		(BX)=文件号	若(CF)=1　失败　(AX)=6 无效文件号
3F	读文件	(AH)=3FH	若(CF)=0　成功　(AX)=实际读的字节数
		(BX)=文件号	若(CF)=1　失败
		(CX)=字节数	则(AX)=5　拒绝存取
		(DS:DX)=缓冲区首址	(AX)=6　无效文件号
40	写文件	(AH)=40H	(AX)=(CX)　成功
		其它同上	否则出错
			此时(AX)=5　拒绝存取
			(AX)=6　无效文件号
41	删除文件	(AH)=41H	若(CF)=0　成功
		(DS:DX)=字符串	若(CF)=1　失败
		(驱动器名.路径名和文件名.扩展名)	则(AX)=2　找不到文件
		地址	(AX)=5　拒绝存取
42	移动文件读写指针	(AH)=42H	若(CF)=0　成功
		(BX)=文件号	若(CF)=1　失败
		(CX:DX)=位移量	则(AX)=1　无效的(AL)
		(AL)=0 从文件开始移	(AX)=6　无效的文件号
		(AL)=1 从当前位置移	
		(AL)=2 从文件结尾移	
43	修改文件属性	(AH)=43H	若(CF)=0　成功　(CX)=文件属性
		(DS:DX)=字符串地址	若(CF)=1　失败
		(AL)为功能码	则(AX)=1　无效功能码
		=0　取文件属性[注]	(AX)=5　文件找不到
		=1　置文件属性	
		(CX)=文件属性	
44	设备文件 I/O 控制	(AH)=44H	
		(BX)=文件号	

功能号	功　　能	入口信息	出口信息
		(AL)=0 读状态	
		(AL)=1 置状态 DX	
		(AL)=2 读数据到缓冲区	
		(DS:DX)=缓冲区首址	
		(CX)=读的字节数	
		(AL)=3 写数据到缓冲区	
		DS,DX,CX 同上	
		(AL)=6 取输入状态	(DX)=状态
		(AL)=7 取输出状态	
45	复制文件号	(AH)=45H	若(CF)=0　成功　(AX)=文件号 2
		(BX)=文件号 1	若(CF)=1　失败
			则(AX)=4 打开文件太多
			(AX)=6 文件号无效
46	强迫复制文件号	(AH)=46H	(CF)=0 成功 (CX)=文件号 1
		(BX)=文件号 1	其它同上
		(CX)=文件号 2	
47	取当前目录路径	(AH)=47H	若(CF)=0 成功 路径全名在所指
		(DL)=驱动器号	内存中
		(DS:SI)=内存地址	若(CF)=1 失败 则(AX)=15
		(64 个字节长)	驱动器名无效
48	分配内存	(AH)=48H	若(CF)=0 成功
		(BX)=申请内存块数	则(AX:0)=分配的内存首址
			若(CF)=1 失败
			则(BX)=最大可用空间块数
			(AX)=7 内存控制块破坏
			(AX)=8 内存不够
49	释放内存	(AH)=49H	若(CF)=0 成功
		(ES:0)=释放内存首址	若(CF)=1 失败
			则(AX)=7 内存控制块破坏
			(AX)=8 内存不够
4A	修改已分配的内存	(AH)=4AH	(CF)=0 成功
		ES 指向已分配段值	(CF)=1 失败
		(BX)=要求内存段数	(BX)=可用最大值
			(AX)=7 内存控制块破坏
			(AX)=8 内存不够
4B	程序的装入	(AH)=4BH	(CF)=0 成功
		(DS:DX)=字符串地址	(CF)=1 失败

功能号	功　能	入口信息	出口信息
		(ES:BX)=参数区首址 (AL)=0 装入执行 (AL)=B 装入不执行	
4C	进程结束,返回操作系统	(AH)=4CH	屏幕显示操作系统提示符 n>
4D	取子进程退出码	(AH)=4DH	若(CF)=0 成功 (AX)=退出码 若(CF)=1 失败 则(AX)=1 无效功能 (AX)=2 文件找不到 (AX)=5 拒绝存取 (AX)=8 内存不够 (AX)=10 环境出错 (AX)=11 格式错
4E	查找第一个文件	(AH)=4EH (DS:DX)=字符串地址 (CX)=文件属性	若(CF)=0 成功　DTA 中有记载信息 (CF)=1 失败 则(AX)=21 文件找不到 (AX)=18 没有文件
4F	查找下一个文件	(AH)=4FH 其它同上	同上
50	DOS 内部使用		
51	DOS 内部使用		
52	DOS 内部使用		
53	DOS 内部使用		
54	取校验开关状态	(AH)=54H	(AL)=00 为 OFF (AL)=01 为 ON
55	DOS 内部使用		
56	改文件名	(AH)=56H (DS:DX)=字符串 1 (ES:DI)=字符串 2 字符串 1、字符串 2 表示驱动器名、路径名与文件名、扩展名 字符串 1 是被改的	若(CF)=0 成功 (CF)=1 失败 则(AX)=2 文件找不到 (AX)=3 路径找不到 (AX)=5 拒绝存取 (AX)=17 不是同一设备
57	置或取文件日期及时间	(AH)=57H (BX)=文件号 (AL)=00 表示读取日期及时间 (AL)=01 表示置日期及时间 (DX:CX)=日期及时间	若(CF)=0 成功 (DX:AX)=日期及时间 若(CF)=1 失败 则(AX)=1 无效的(AL) (AX)=6 无效的文件号
58	置或取分配策略码	(AH)=58H	若(CF)=0 成功 (AX)=策略码

注 1:文件属性占一个字节,其含义如下:

7	6	5	4	3	2	1	0
×	×	更改位	子目录	卷标位	系统	隐含	只读

其中有些属性可组合使用,对具有隐含、系统和子目录属性的文件,不能用一般的目录操作检索。

附录E BIOS中断

INT	AH	功能	调用参数	返回参数
10	0	设置显示方式	AL =00 40×25 黑白方式	
			=01 40×25 彩色方式	
			=02 80×25 黑白方式	
			=03 80×25 彩色方式	
			=04 320×200 彩色图形方式	
			=05 320×200 黑白图形方式	
			=06 640×200 黑白图形方式	
			=07 80×25 单色文本方式	
			=08 160×200 16 色图形(PCjr)	
			=09 320×200 16 色图形(PCjr)	
			=0A 640×200 16 色图形(PCjr)	
			=0B 保留 (EGA)	
			=0C 保留 (EGA)	
			=0D 320×200 彩色图形(EGA)	
			=0E 640×200 彩色图形(EGA)	
			=0F 640×350 黑白图形(EGA)	
			=10 640×350 彩色图形(EGA)	
			=11 640×480 单色图形(EGA)	
			=12 640×480 16 色图形(EGA)	
			=13 320×200 256 色图形(EGA)	
			=40 80×30 彩色文本(CEG400)	
			=41 80×50 彩色文本(CEG400)	
			=42 640×400 彩色文本(CEG400)	
10	1	置光标类型	(CH) 0-3=光标起始行	
			(CL) 0-3=光标结束行	
10	2	置光标位置	BH=页号	
			DH,DL=行,列	
10	3	读光标位置	BH=页号	CH=光标起始行
				DH,DL=行,列
10	4	读光笔位置		AH=0 光笔未触发
				=1 光笔触发
				CH=象素行
				BX=象素列
				DH=字符行
				DL=字符列

INT	AH	功能	调用参数	返回参数
			DH,DL=起始行,列	
			BH=页号	
			AL=0,BL=属性	
			串:char,char,....	光标返回起始位置
			AL=1,BL=属性	光标跟随移动
			串:char,char,....	
			AL=2	光标返回起始位
			串:char,char,char,char,...	
			AL=3	光标跟随移动
			串:char,char,char,char,...	
11		设备检验		AX=返回值
				bit0=1,配有磁盘
				bit1=1,80287 协处理
				bit4,5=01,40×25BW(彩色板)
				=10,80×25BW(彩色板)
				=11,80×25BW(黑白板)
				bit6,7=软盘驱动器号
				bit9,10,11=RS-232 板号
				bit12=游戏适配器
				bit13=串行打印机
				bit14,15=打印机号
12		测定存储容量		AX=字节数(Kb)
13	0	软盘系统复位		
13	1	读软盘状态		AL=状态字节
13	2	读磁盘	AL=扇区数	读成功:AH=0
			CH,CL=磁道号,扇区号	AL=读取的扇区数
			DH,DL=磁头号,驱动器号	读失败:
			ES:BX=数据缓冲区地址	AH=出错代码
13	3	写磁盘	同上写	成功:AH=0
				AL=写入的扇区数
				写失败:
				AH=出错代码
13	4	检验磁盘扇区	同上(ES:BX 不设置)	成功:AH=0
				AL=检验的扇区数
				失败:AH=出错代码
13	5	格式化盘磁道	ES:BX=磁道地址	成功:AH=0

INT	AH	功能	调用参数	返回参数
				失败:AH=出错代码
14	0	初始化串行通讯口	AL=初始化参数 DX=通讯口号(0,1)	AH=通讯解调器状态 AL=调制解调器状态
14	1	向串行通讯口写字符	AL=字符 DX=通讯口号(0,1)	写成功:(AH)7=0 写失败:(AH)7=1 (AH)0-6=通讯口状态
14	2	从串行通讯口取字符	DX=通讯口号(0,1)	读成功:(AH)7=0 (AL)=字符 读失败:(AH)7=1 (AH)0-6=通讯口状态
14	3	取通讯口状态	DX=通讯口号(0,1)	AH=通讯口状态 AL=调制调解器状态
15	0	启动盒式磁带马达		
15	1	停止盒式磁带马达		
15	2	磁带分块读	ES:BX=数据传输区地址 CX=字节数	AH=状态字节 AH=00 读成功 =01 冗余检验错 =02 无数据传输 =04 无引导 =80 非法命令
15	3	磁带分块写	DS:BX=数据传输区地址 CX=字节数	AH=状态字节 (同上)
16	0	从键盘读字符		AL=字符码 AH=扫描码
16	1	读键盘缓冲区字符		ZF=0 AL=字符码 AH=扫描码 ZF=1 缓冲区空
16	2	取键盘状态字节		AL=键盘状态字节
17	0	打印字符,回送状态字节	AL=字符 DX=打印机号	AH=打印机状态字节
17	1	初始化打印机回送状态字节	DX=打印机号	AH=打印机状态字节
17	2	取状态字节	DX=打印机号	AH=打印机状态字节

INT	AH	功能	调用参数	返回参数
1A	0	读时钟		CH:CL=时:分 DH:DL=秒:1/100秒
1A	1	置时钟	CH:CL=时:分 DH:DL=秒:1/100秒	
1A	2	读实时钟 (适用AT)		CH:CL=时:分(BCD) DH:DL=秒:1/100秒(BCD)
1A	6	置报警时间 (适用AT)	CH:CL=时:分(BCD) DH:DL－秒:1/100秒(BCD)	
1A	7	清除报警 (适用AT)		

主要参考资料

1. 何希才、尤　克、姜余祥、林　丽　编著
《80486微型计算机的原理及应用》　人民邮电出版社 1994.10
2. 周明德、白晓笛　编著
《微型计算机从8086到80386》　清华大学出版社　1992.3
3. 曾家智、向世清　编著
《微型计算机系统与接口》　电子科技大学出版社　1992.7
4. 孟昭光、李维星　编著
《高档微机组成原理及接口技术》　学苑出版社　1993.11
5. 曲伯涛　编著
《8086到80486微型计算机系统原理与接口》　大连理工大学出版社　1994.12
6. Intel《Micro System Components Handbook》　October.1983
7. Thom Hoarg　著
《PC硬软件技术资料大全》(计　帆译)清华大学出版社　1991.2
8 戴梅萼　编著
《微型计算机技术及应用》　清华大学出版社　1991.11
9. 周　细　等编
《微型计算机及其应用》　华中理工大学出版社　1992.5
10. 舒贞权　等编
《微型计算机原理》　西安交通大学出版社　1993.12
11. 幸云辉　编著
《16位微型计算机原理与应用》　北京邮电学院出版社　1992.2
12. 潘名莲　等编
《微型计算原理》　电子工业出版社　1994.12
13. 杨素行　等编
《微型计算机系统原理及应用》　清华大学出版社　1995.9
14. 侯伯亨、李伯成　编著
《十六位微型计算机原理及接口技术》　西安电子科技大学出版社　1992.2
15.《The 8086 Family User's Manual》　1979 Intel Corp
16. Walter A. Triebel, AvtarSingh.
《16-Bit Microprocessors Architecture Software and Interface techniques》Englewood Cliffs, Prentice-Hall, 1985.
17. 周佩玲、吴耿锋、万炳奎　编著
《16位微型计算机原理、接口及其应用》　中国科学技术大学出版社　1995.8
18. 白中英　编著
《计算机组成原理》　科学出版社　1994.6
19. 张瑞庭　主编
《微型计算机原理》　人民邮电出版社　1995.2
20. 易仲芳　主编
《80X86微型计算机原理及应用》　电子工业出版社　1995.4
21. 杨振生、单　洪　编著
《8086/8088汇编语言程序设计》　中国科学技术大学出版社　1994